



DEITEL®

Il linguaggio C

Fondamenti e tecniche di programmazione

Nona edizione

Paul J. Deitel, Harvey M. Deitel

MyLab Codice per accedere
alla piattaforma

CODICE STUDENTE MONOUSO

ISBN 9788891808236B

RSLJLE-BEGAD-IVORY-WRITE-ORACY-FLEES

Attivabile dal 05/09/21 fino al 31/12/26. Durata 18 mesi LR316173B



Pearson

Il linguaggio C

© 2022 Pearson Italia, Milano - Torino

*Authorized translation from the English language edition, entitled C How to Program, 9th Edition, by Paul Deitel and Harvey Deitel,
published by Pearson Education, Inc. publishing as Pearson. Copyright © 2022.*

*All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical,
including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.*

Italian language edition published by Pearson Italia S.p.A., Copyright © 2022.

Per i passi antologici, per le citazioni, per le riproduzioni grafiche, cartografiche e fotografiche appartenenti alla proprietà di terzi, inseriti in quest'opera, l'editore è a disposizione degli avenuti diritto non potuti reperire nonché per eventuali non volute omissioni e/o errori di attribuzione nei riferimenti.

È vietata la riproduzione, anche parziale o ad uso interno didattico, con qualsiasi mezzo, non autorizzata.
Le fotocopie per uso personale del lettore possono essere effettuate nei limiti del 15% di ciascun volume dietro pagamento alla SIAE del compenso previsto dall'art. 68, commi 4 e 5, della legge 22 aprile 1941, n. 633.
Le riproduzioni effettuate per finalità di carattere professionale, economico o commerciale o comunque per uso diverso da quello personale possono essere effettuate a seguito di specifica autorizzazione rilasciata da CLEARED, Corso di Porta Romana 108, 20122 Milano, e-mail autorizzazioni@clearedi.org e sito web www.clearedi.org.

I nostri libri sono ecosostenibili: la carta è prodotta sostenendo il ciclo naturale e per ogni albero tagliato ne viene piantato un altro; il cellofan è realizzato con plastiche da recupero ambientale o riciclate; gli inchiostri sono naturali e atossici; i libri sono prodotti in Italia e l'impatto del trasporto è ridotto al minimo.

Traduzione e realizzazione editoriale: Giulia Maselli e Maria Mantero

Grafica di copertina: Simone Tartaglia

Immagine di copertina: Willyam Bradberry/Shutterstock

Stampa: Arti Grafiche Battaia (Zibido San Giacomo - MI)

Tutti i marchi citati nel testo sono di proprietà dei loro detentori.

9788891906236

Printed in Italy

9^a edizione: settembre 2022

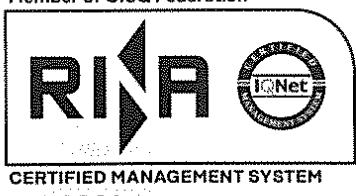
Ristampa
00 01 02 03 04

Anno
22 23 24 25 26

LIBRI DI TESTO E SUPPORTI DIDATTICI

Il sistema di gestione per la qualità della Casa Editrice è certificato in conformità alla norma UNI EN ISO 9001:2015 per l'attività di progettazione, realizzazione e commercializzazione di: • prodotti editoriali scolastici, dizionari lessicografici, prodotti per l'editoria di varia ed università • materiali didattici multimediali off-line • corsi di formazione e specializzazione in aula, a distanza, e-learning.

Member of CISQ Federation



*In memoria di Dennis Ritchie,
creatore del linguaggio di programmazione C
e co-creatore del sistema operativo UNIX.*

Paul and Harvey Deitel

Sommario

Prefazione all'edizione italiana	XXI
Prefazione	XXIII
Prima di cominciare	XLV
Pearson MyLab	XLIX

Capitolo	Introduzione ai computer e al linguaggio C	1
1.1	Introduzione	1
1.2	Hardware e software	2
1.2.1	Legge di Moore	3
1.2.2	Organizzazione dei computer	3
1.3	Gerarchia dei dati	5
1.4	Linguaggi macchina, assembly e di alto livello	8
1.5	Sistemi operativi	10
1.6	Il linguaggio di programmazione C	12
1.7	Libreria Standard del C e librerie open source	14
1.8	Altri linguaggi di programmazione popolari	15
1.9	Un tipico ambiente di sviluppo per la programmazione in C	16
1.9.1	Fase 1: creazione di un programma	16
1.9.2	Fasi 2 e 3: preelaborazione e compilazione di un programma in C	17
1.9.3	Fase 4: linking	17
1.9.4	Fase 5: loading	18
1.9.5	Fase 6: esecuzione	18
1.9.6	Problemi che possono presentarsi al momento dell'esecuzione	19
1.9.7	Gli stream standard input, standard output e standard error	19
1.10	Test: eseguire un'applicazione in C negli ambienti Windows, Linux e macOS	19
1.10.1	Compilare ed eseguire un'applicazione in C con Visual Studio 2019 Community Edition su Windows 10	20

1.10.2	Compilare ed eseguire un'applicazione in C con Xcode su macOS	24
1.10.3	Compilare ed eseguire un'applicazione in C con GNU gcc su Linux	26
1.10.4	Compilare ed eseguire un'applicazione in C usando un contenitore Docker della GCC in esecuzione nativa su Windows 10, macOS o Linux	28
1.11	Internet, World Wide Web, cloud e IoT	29
1.11.1	Internet: una rete di reti	30
1.11.2	World Wide Web: come rendere Internet user-friendly	30
1.11.3	Cloud	30
1.11.4	Internet delle cose	31
1.12	Tecnologie software	32
1.13	Quanto sono grandi i big data?	32
1.13.1	Analisi dei big data	36
1.13.2	Big data e data science stanno facendo la differenza: casi d'uso	37
1.14	Caso pratico: un'applicazione mobile di big data	38
1.15	Intelligenza artificiale, all'intersezione tra informatica e data science	39
1.16	Riepilogo	41
Capitolo 2	Introduzione alla programmazione nel linguaggio C	45
2.1	Introduzione	45
2.2	Un semplice programma in C: stampare una riga di testo	45
2.3	Un altro semplice programma in C: sommare due interi	49
2.4	Concetti relativi alla memoria	52
2.5	Aritmetica in C	53
2.6	Decisioni: operatori di uguaglianza e relazionali	57
2.7	Programmazione sicura in C	60
2.8	Riepilogo	62
Capitolo 3	Sviluppo di un programma strutturato	71
3.1	Introduzione	71
3.2	Algoritmi	71
3.3	Pseudocodice	72
3.4	Strutture di controllo	73
3.5	Istruzione di selezione if	75
3.6	Istruzione di selezione if...else	76
3.7	Istruzione di iterazione while	80
3.8	Formulare algoritmi, caso pratico 1: iterazione controllata da contatore	81

3.9	Formulare algoritmi con affinamento graduale top-down, caso pratico 2: iterazione controllata da sentinella	83
3.10	Formulare algoritmi con affinamento graduale top-down, caso pratico 3: istruzioni di controllo annidate	89
3.11	Operatori di assegnazione	93
3.12	Operatori di incremento e di decremento	93
3.13	Programmazione sicura in C	96
3.14	Riepilogo	97

Capitolo	4	Controllo nei programmi	113
-----------------	----------	--------------------------------	------------

4.1	Introduzione	113
4.2	Aspetti essenziali dell'iterazione	113
4.3	Iterazione controllata da contatore	114
4.4	Istruzione di iterazione for	115
4.5	Esempi di uso dell'istruzione for	118
4.6	Istruzione di selezione multipla switch	122
4.7	Istruzione di iterazione do...while	127
4.8	Istruzioni break e continue	129
4.9	Operatori logici	131
4.10	Confondere gli operatori di uguaglianza (==) e di assegnazione (=)	134
4.11	Sintesi della programmazione strutturata	135
4.12	Programmazione sicura in C	140
4.13	Riepilogo	141

Capitolo	5	Funzioni	155
-----------------	----------	-----------------	------------

5.1	Introduzione	155
5.2	Modularizzazione dei programmi in C	156
5.3	Funzioni della libreria math	157
5.4	Funzioni	158
5.5	Definizioni di funzioni	159
5.5.1	Funzione square	159
5.5.2	Funzione maximum	161
5.6	Prototipi di funzioni: uno sguardo più approfondito	163
5.7	Pila delle chiamate delle funzioni e record di attivazione	165
5.8	File di intestazione	169
5.9	Passare gli argomenti per valore e per riferimento	170
5.10	Generazione di numeri casuali	171

5.11	Caso pratico sulla simulazione di numeri casuali: creare un gioco da casinò	175
5.12	Classi di memoria	178
5.13	Regole per il campo d'azione	180
5.14	Ricorsione	183
5.15	Esempio di utilizzo della ricorsione: la serie di Fibonacci	187
5.16	La ricorsione rispetto all'iterazione	189
5.17	Programmazione sicura in C – Generazione sicura di numeri casuali	192
5.18	Riepilogo	193
Capitolo 6	Array	211
6.1	Introduzione	211
6.2	Array	212
6.3	Definire gli array	213
6.4	Esempi di array	214
6.4.1	Definire un array e usare un ciclo per impostare i valori degli elementi di un array	214
6.4.2	Inizializzare un array in una definizione con una lista di inizializzatori	215
6.4.3	Specificare la dimensione di un array con una costante simbolica e inizializzare gli elementi dell'array con espressioni da calcolare	215
6.4.4	Sommare gli elementi di un array	217
6.4.5	Usare array per riassumere i risultati di un sondaggio	217
6.4.6	Rappresentare con grafici a barre i valori degli elementi di un array	219
6.4.7	Lanciare un dado 60.000.000 di volte e riepilogare i risultati in un array	220
6.5	Uso di array di caratteri per memorizzare e manipolare stringhe	221
6.5.1	Inizializzazione di un array di caratteri con una stringa	221
6.5.2	Inizializzazione di un array di caratteri con una lista di inizializzatori di caratteri	221
6.5.3	Accesso ai caratteri di una stringa	222
6.5.4	Inserimento in un array di caratteri	222
6.5.5	Stampa di un array di caratteri che rappresenta una stringa	222
6.5.6	Illustrazione di un array di caratteri	222
6.6	Array locali statici e array locali automatici	224
6.7	Passare gli array alle funzioni	226
6.8	Ordinamento di array	230
6.9	Caso pratico di introduzione alla data science: analisi dei dati di un sondaggio	232
6.10	Ricerca in array	237

6.10.1	Effettuare una ricerca in un array con la ricerca lineare	237
6.10.2	Effettuare una ricerca in un array con la ricerca binaria	238
6.11	Array multidimensionali	242
6.11.1	Illustrare un array bidimensionale	243
6.11.2	Inizializzare un array con due indici	243
6.11.3	Impostare gli elementi in una riga	245
6.11.4	Calcolare il totale degli elementi in un array bidimensionale	245
6.11.5	Manipolazioni di array bidimensionali	245
6.12	Array di lunghezza variabile	249
6.13	Programmazione sicura in C	252
6.14	Riepilogo	254

Capitolo 7	Puntatori	271
7.1	Introduzione	271
7.2	Definizioni e inizializzazione di variabili puntatore	271
7.3	Operatori per i puntatori	273
7.4	Passare argomenti a funzioni per riferimento	275
7.5	Uso del qualificatore const con i puntatori	280
7.5.1	Conversione di una stringa in maiuscolo usando un puntatore non costante a dati non costanti	280
7.5.2	Stampa di una stringa un carattere alla volta usando un puntatore non costante a dati costanti	281
7.5.3	Tentativo di modificare un puntatore costante a dati non costanti	283
7.5.4	Tentativo di modificare un puntatore costante a dati costanti	283
7.6	Bubble sort che utilizza il passaggio per riferimento	284
7.7	Operatore sizeof	287
7.8	Espressioni con puntatori e aritmetica dei puntatori	290
7.8.1	Operatori per l'aritmetica dei puntatori	290
7.8.2	Indirizzare un puntatore a un array	290
7.8.3	Aggiungere un intero a un puntatore	290
7.8.4	Sottrarre un intero da un puntatore	291
7.8.5	Incrementare e decrementare un puntatore	291
7.8.6	Sottrarre un puntatore da un altro	291
7.8.7	Assegnare puntatori ad altri puntatori	291
7.8.8	Puntatore a void	292
7.8.9	Confrontare i puntatori	292
7.9	Relazioni tra puntatori e array	292
7.9.1	Notazione puntatore/offset	292
7.9.2	Notazione puntatore/indice	293

7.9.3	Non è possibile modificare il nome di un array con l'aritmetica dei puntatori	293
7.9.4	Dimostrare indicizzazione e offset con un puntatore	293
7.9.5	Copiare stringhe con array e puntatori	295
7.10	Array di puntatori	296
7.11	Caso pratico di simulazione di numeri casuali: mescolare e distribuire le carte	297
7.12	Puntatori a funzioni	302
7.12.1	Ordinamento crescente o decrescente	302
7.12.2	Uso dei puntatori a funzioni per realizzare un sistema guidato da menu	305
7.13	Programmazione sicura in C	306
7.14	Riepilogo	307

Capitolo 8	Caratteri e stringhe	341
8.1	Introduzione	341
8.2	Nozioni fondamentali su stringhe e caratteri	341
8.3	Libreria per il trattamento dei caratteri	343
8.3.1	Funzioni isdigit, isalpha, isalnum e isxdigit	344
8.3.2	Funzioni islower, isupper, tolower e toupper	345
8.3.3	Funzioni isspace, iscntrl, ispunct, isprint e isgraph	346
8.4	Funzioni di conversione di stringhe	348
8.4.1	Funzione strtod	348
8.4.2	Funzione strtol	349
8.4.3	Funzione strtoul	350
8.5	Funzioni della libreria standard di input/output	351
8.5.1	Funzioni fgets e putchar	351
8.5.2	Funzione getchar	352
8.5.3	Funzione sprintf	353
8.5.4	Funzione sscanf	354
8.6	Funzioni per la manipolazione di stringhe della libreria per il trattamento delle stringhe	355
8.6.1	Funzioni strcpy e strncpy	355
8.6.2	Funzioni strcat e strncat	356
8.7	Funzioni di confronto della libreria per il trattamento delle stringhe	357
8.8	Funzioni per la ricerca della libreria per il trattamento delle stringhe	359
8.8.1	Funzione strchr	360
8.8.2	Funzione strcspn	361

8.8.3	Funzione <code>strpbrk</code>	361
8.8.4	Funzione <code>strrchr</code>	362
8.8.5	Funzione <code>strspn</code>	362
8.8.6	Funzione <code>strstr</code>	363
8.8.7	Funzione <code>strtok</code>	363
8.9	Funzioni di gestione della memoria della libreria per il trattamento delle stringhe	365
8.9.1	Funzione <code>memcpy</code>	366
8.9.2	Funzione <code>memmove</code>	366
8.9.3	Funzione <code>memcmp</code>	367
8.9.4	Funzione <code>memchr</code>	367
8.9.5	Funzione <code>memset</code>	368
8.10	Altre funzioni della libreria per il trattamento delle stringhe	369
8.10.1	Funzione <code>strerror</code>	369
8.10.2	Funzione <code>strlen</code>	369
8.11	Programmazione sicura in C	370
8.12	Riepilogo	371
Capitolo 9	Input/output formattato	395
9.1	Introduzione	395
9.2	Stream	395
9.3	Formattazione dell'output con <code>printf</code>	396
9.4	Stampa di interi	397
9.5	Stampa di numeri in virgola mobile	398
9.5.1	Specificatori di conversione e, E e f	399
9.5.2	Specificatori di conversione g e G	399
9.5.3	Dimostrare gli specificatori di conversione in virgola mobile	399
9.6	Stampa di stringhe e caratteri	400
9.7	Altri specificatori di conversione	401
9.8	Stampare con larghezza di campo e precisione	402
9.8.1	Larghezze di campo per interi	402
9.8.2	Precisione per interi, numeri in virgola mobile e stringhe	403
9.8.3	Combinare larghezze di campo e precisioni	404
9.9	Flag di formato di <code>printf</code>	405
9.9.1	Allineamento a destra e a sinistra	405
9.9.2	Stampare numeri positivi e negativi con e senza il flag +	406
9.9.3	Usare il flag spazio	406
9.9.4	Usare il flag #	407
9.9.5	Usare il flag 0	407

9.10	Stampa di letterali e sequenze di escape	408
9.11	Input formattato con scanf	409
9.11.1	Sintassi di scanf	409
9.11.2	Specificatori di conversione di scanf	410
9.11.3	Leggere interi	411
9.11.4	Leggere numeri in virgola mobile	411
9.11.5	Leggere caratteri e stringhe	412
9.11.6	Usare insiemi di scansione	412
9.11.7	Usare larghezze di campo	413
9.11.8	Tralasciare caratteri in uno stream di input	414
9.12	Programmazione sicura in C	415
9.13	Riepilogo	416
Capitolo 10	Strutture, unioni, manipolazione di bit ed enumerazioni	423
10.1	Introduzione	423
10.2	Definizione di strutture	424
10.2.1	Strutture autoreferenziali	424
10.2.2	Definizione di variabili di tipi di struttura	425
10.2.3	Etichette delle strutture	425
10.2.4	Operazioni che si possono eseguire sulle strutture	425
10.3	Inizializzazione di strutture	426
10.4	Accesso ai membri delle strutture con . e ->	427
10.5	Uso delle strutture con le funzioni	429
10.6	typedef	429
10.7	Caso pratico di simulazione di numeri casuali: mescolamento e distribuzione di carte ad alte prestazioni	430
10.8	Unioni	433
10.8.1	Dichiarazione di unioni	433
10.8.2	Operazioni consentite sulle unioni	434
10.8.3	Inizializzare unioni nelle dichiarazioni	434
10.8.4	Illustrazione delle unioni	434
10.9	Operatori bit a bit	435
10.9.1	Stampa di un intero senza segno nella sua rappresentazione in bit	436
10.9.2	Rendere la funzione displayBits più generica e portabile	437
10.9.3	Uso degli operatori bit a bit AND, OR inclusivo, OR esclusivo e complemento	438
10.9.4	Uso degli operatori bit a bit di spostamento a sinistra e a destra	441
10.9.5	Operatori di assegnazione bit a bit	442
10.10	Campi di bit	444
10.10.1	Definire campi di bit	444

10.10.2	Usare campi di bit per rappresentare i membri face, suit e color di una carta	444
10.10.3	Campi di bit anonimi	446
10.11	Costanti di enumerazione	447
10.12	Strutture e unioni anonime	449
10.13	Programmazione sicura in C	449
10.14	Riepilogo	450
Capitolo 11	Elaborazione di file	475
11.1	Introduzione	475
11.2	File e stream	475
11.3	Creazione di un file ad accesso sequenziale	476
11.3.1	Puntatore a un FILE	477
11.3.2	Usare fopen per aprire il file	478
11.3.3	Usare feof per controllare l'indicatore di end-of-file	478
11.3.4	Usare fprintf per scrivere su un file	478
11.3.5	Usare fclose per chiudere il file	478
11.3.6	Modalità di apertura dei file	479
11.4	Lettura di dati da un file ad accesso sequenziale	481
11.4.1	Reimpostare il puntatore di posizione del file	482
11.4.2	Programma di interrogazione per il credito	482
11.5	File ad accesso casuale	485
11.6	Creazione di un file ad accesso casuale	486
11.7	Scrittura di dati in maniera casuale su un file ad accesso casuale	488
11.7.1	Collocare il puntatore di posizione del file con fseek	490
11.7.2	Controllo sugli errori	491
11.8	Lettura di dati da un file ad accesso casuale	491
11.9	Caso pratico: sistema per l'elaborazione di transazioni	492
11.10	Programmazione sicura in C	498
11.11	Riepilogo	499
Capitolo 12	Strutture di dati	523
12.1	Introduzione	523
12.2	Strutture autoreferenziali	524
12.3	Gestione dinamica della memoria	525
12.4	Liste collegate	526
12.4.1	Funzione insert	530
12.4.2	Funzione delete	532
12.4.3	Funzioni isEmpty e printList	533

12.5 Pile	534
12.5.1 Funzione push	538
12.5.2 Funzione pop	538
12.5.3 Applicazioni delle pile	539
12.6 Code	540
12.6.1 Funzione enqueue	544
12.6.2 Funzione dequeue	545
12.7 Alberi	546
12.7.1 Funzione insertNode	549
12.7.2 Attraversamenti: funzioni inOrder, preOrder e postOrder	549
12.7.3 Eliminazione dei duplicati	550
12.7.4 Ricerca in un albero binario	551
12.7.5 Altre operazioni su alberi binari	551
12.8 Programmazione sicura in C	552
12.9 Riepilogo	552
Capitolo 13 Pensare come un informatico: algoritmi di ordinamento e O grande	577
13.1 Introduzione	577
13.2 Efficienza degli algoritmi: O grande	578
13.2.1 Algoritmi $O(1)$	578
13.2.2 Algoritmi $O(n)$	578
13.2.3 Algoritmi $O(n^2)$	578
13.3 Ordinamento per selezione	579
13.3.1 Implementazione dell'ordinamento per selezione	580
13.3.2 Efficienza dell'ordinamento per selezione	583
13.4 Ordinamento per inserzione	583
13.4.1 Implementazione dell'ordinamento per inserzione	584
13.4.2 Efficienza dell'ordinamento per inserzione	586
13.5 Caso pratico: visualizzazione dell'algoritmo di ordinamento per fusione ad alte prestazioni	587
13.5.1 Implementazione dell'ordinamento per fusione	587
13.5.2 Efficienza dell'ordinamento per fusione	592
13.5.3 Riepilogo di valori della notazione O grande di alcuni algoritmi	592
13.6 Riepilogo	594
Capitolo 14 Preprocessore	599
14.1 Introduzione	599
14.2 Direttiva per il preprocessore #include	600

14.3	Direttiva per il preprocessore #define: costanti simboliche	601
14.4	Direttiva per il preprocessore #define: macro	602
14.4.1	Macro con un argomento	602
14.4.2	Macro con due argomenti	603
14.4.3	Carattere di continuazione per macro	603
14.4.4	Direttiva per il preprocessore #undef	603
14.4.5	Macro della Libreria Standard	603
14.4.6	Evitare espressioni con effetti secondari nelle macro	603
14.5	Compilazione condizionale	604
14.5.1	Direttiva per il preprocessore #if...#endif	604
14.5.2	Commentare porzioni di codice con #if...#endif	604
14.5.3	Compilazione condizionale e codice di debugging	605
14.6	Direttive per il preprocessore #error e #pragma	606
14.7	Operatori # e ##	606
14.8	Numeri di riga	607
14.9	Costanti simboliche predefinite	607
14.10	Asserzioni	608
14.11	Programmazione sicura in C	608
14.12	Riepilogo	609
Capitolo 15	Altri argomenti	613
15.1	Introduzione	613
15.2	Liste di argomenti di lunghezza variabile	613
15.3	Uso degli argomenti della riga di comando	616
15.4	Compilazione di programmi con più file sorgente	617
15.4.1	Dichiarazioni extern per variabili globali in altri file	617
15.4.2	Prototipi di funzioni	618
15.4.3	Restringere il campo d'azione con static	618
15.5	Terminazione di programmi con exit e atexit	619
15.6	Suffissi per letterali interi e in virgola mobile	620
15.7	Gestione dei segnali	621
15.8	Allocazione dinamica della memoria: funzioni malloc e realloc	624
15.9	goto: salto non condizionato	625
15.10	Riepilogo	626
Appendice A	Tabella di precedenza degli operatori	631
Appendice B	Insieme di caratteri ASCII	633

Appendice C Multithreading/multicore e altri argomenti di C18/C11/C99	635
C.1 Introduzione	635
C.2 File di intestazione aggiunti nel C99	636
C.3 Inizializzatori designati e letterali composti	636
C.4 Il tipo <code>bool</code>	638
C.5 Numeri complessi	639
C.6 Macro con liste di argomenti di lunghezza variabile	640
C.7 Altre caratteristiche del C99	641
C.7.1 Limiti minimi delle risorse per i compilatori	641
C.7.2 La parola chiave <code>restrict</code>	641
C.7.3 Divisione intera affidabile	641
C.7.4 Membri array flessibili	642
C.7.5 Tipo <code>math</code> generico	642
C.7.6 Funzioni inline	642
C.7.7 Identificatore predefinito <code>__func__</code>	643
C.7.8 Macro <code>va_copy</code>	643
C.8 Caratteristiche di C11/C18	643
C.8.1 File di intestazione di C11/C18	643
C.8.2 Funzione <code>quick_exit</code>	643
C.8.3 Supporto Unicode®	644
C.8.4 Specificatore di funzione <code>_Noreturn</code>	644
C.8.5 Espressioni di tipo generico	644
C.8.6 Annex L: analizzabilità e comportamento indefinito	644
C.8.7 Controllo dell'allineamento in memoria	645
C.8.8 Asserzioni statiche	645
C.8.9 Tipi in virgola mobile	645
C.9 Caso pratico: prestazioni con sistemi multithreading e multicore	646
C.9.1 Esempio: esecuzione sequenziale di due attività di calcolo intensivo	648
C.9.2 Esempio: esecuzione multithreaded di due attività di calcolo intensivo	650
C.9.3 Altre caratteristiche del multithreading	654
Appendice D Introduzione ai principi della programmazione orientata agli oggetti	655
D.1 Introduzione	655
D.2 Linguaggi di programmazione orientata agli oggetti	655
D.3 L'automobile come oggetto	655
D.4 Metodi e classi	656
D.5 Istanziazione	656
D.6 Riutilizzo	656

D.7	Messaggi e chiamate di metodo	656
D.8	Attributi e variabili di istanza	656
D.9	Ereditarietà	657
D.10	Analisi e progettazione orientate agli oggetti (OOAD)	657

Appendice E	Sistemi di numerazione	ONLINE
--------------------	-------------------------------	---------------

Appendice F	Utilizzo del debugger Visual Studio	ONLINE
--------------------	--	---------------

Appendice G	Utilizzo del debugger GNU	ONLINE
--------------------	----------------------------------	---------------

Appendice H	Utilizzo del debugger di Xcode	ONLINE
--------------------	---------------------------------------	---------------

Indice analitico	663
-------------------------	------------

Prefazione all'edizione italiana

La nona edizione di questo testo esce con una traduzione molto fedele all'originale. Ho seguito abbastanza da vicino l'evoluzione del testo come docente di un insegnamento di Fondamenti di Programmazione al I anno del corso di laurea in Ingegneria Informatica presso l'Università degli Studi di Palermo, avendo adottato negli anni questo volume come libro di testo. Mi riferisco a un primo corso di programmazione che gli studenti affrontano parallelamente a un corso di architetture di elaborazione. Il corso verte principalmente sugli aspetti metodologici della programmazione con un taglio orientato all'effettivo sviluppo di applicazioni. Di grande aiuto al riguardo è l'approccio proposto nel libro basato su codice funzionante e analisi dettagliata. Del libro ho curato la traduzione in italiano della settima edizione. Questa usciva proprio nel momento in cui veniva definito lo standard C11, successivo allo standard C99. Da qui l'esigenza, da parte degli autori, di ulteriori edizioni, anche ravvicinate, che tenessero pienamente conto di questo nuovo standard e non trascurassero il successivo C18.

La nona edizione è stata soggetta a una profonda revisione, pur essendo stato mantenuto lo stesso approccio didattico e metodologico. Il risultato è un volume completo sul linguaggio C, adatto sia a chi deve imparare a programmare sia a chi ha già esperienza di programmazione. Proprio per costoro il libro costituisce un valido e completo riferimento. È stato aggiornato specialmente il codice C degli esempi, e anche i paragrafi sono stati riorganizzati in modo più puntuale, con un'articolazione in sottoparagrafi corredati da esercizi di autovalutazione, al fine di permettere un più agevole accesso agli argomenti e uno studio più interattivo e personalizzato. Sono stati inseriti ulteriori chiarimenti e discussioni sulle soluzioni presentate. Sono stati anche ampliati gli esercizi, ma soprattutto sono stati aggiunti nuovi esempi e progetti dettagliati riguardanti tematiche molto attuali, come la simulazione di sistemi, la programmazione di sistemi embedded, la realizzazione di videogiochi, la visualizzazione e la grafica 2D e 3D, l'intelligenza artificiale e la robotica, l'analisi dei dati, la cybersecurity, i servizi web e il multithreading per i sistemi multicore.

Dopo un'ampia trattazione introduttiva sui sistemi informatici e sullo stato dell'arte delle tecnologie informatiche, nel testo l'approccio metodologico si snoda partendo dalle tecniche di programmazione strutturata e procedendo con i costrutti del linguaggio che la supportano, fino a mostrare, con numerosi e corposi esempi ed esercizi, come costruire effettivamente sistemi software anche complessi tramite l'approccio funzionale. Nel corso della trattazione vengono affrontate ampiamente e in modo operativo le tematiche relative agli algoritmi e alle strutture di dati, fondamentali per un approccio professionale alla programmazione.

L'opera si rivolge sia a coloro che operano già in ambito tecnico e scientifico, per i quali può essere un valido riferimento, sia a coloro che vogliono intraprendere un percorso serio e rigoroso che li porti a saper programmare. Il testo è, pertanto, particolarmente adatto a essere utilizzato in corsi universitari riguardanti la programmazione, sia introduttivi che avanzati. A tal proposito, la prefazione indica diversi possibili percorsi didattici.

In merito poi alla collocazione della programmazione in linguaggio C nell'ambito degli insegnamenti universitari dei corsi di laurea in informatica e in ingegneria informatica, possiamo dire che tale problematica va inquadrata nell'ampio dibattito relativo al primo linguaggio con cui introdurre la programmazione. Dopo l'esplosione del Web come fenomeno sociale e l'evoluzione tecnologica che ha portato al cloud, ai big data, alle nuove frontiere dell'intelligenza artificiale, è abbastanza diffusa l'opinione di iniziare il percorso formativo con un approccio più di tipo sistemistico e basato sull'astrazione, attraverso l'utilizzo di linguaggi come Java, e più recentemente Python, che introducono subito anche la metodologia a oggetti e sono supportati da robuste librerie per svariate applicazioni. Personalmente ritengo più opportuno per un corso di laurea in informatica o in ingegneria informatica un percorso che parte dalla dettagliata conoscenza dell'architettura hardware e pro-

cede a tappe verso modelli più astratti e sistemistici. Lo studente che vuole diventare un professionista dell'informatica deve ben sapere cosa sta alla base dei costrutti più astratti. Il linguaggio C è il linguaggio giusto da cui partire, perché, pur essendo un linguaggio di alto livello, è direttamente comprensibile in riferimento alla macchina su cui girano i programmi. Esso permette di introdurre facilmente gli ulteriori livelli di astrazione, che si possono poi, in modo più appropriato, sviluppare nel contesto di altri linguaggi. La conoscenza della programmazione in C è anche propedeutica per i corsi successivi che riguardano il software di sistema, proprio perché è il linguaggio che nasce per tale scopo. È infatti il linguaggio con cui è stato sviluppato il sistema operativo UNIX. È sicuramente quello più adatto agli aspetti legati alle prestazioni. Permette infatti la scrittura di programmi compatti, efficienti ed estremamente portabili. A questo riguardo, nonostante diverse standardizzazioni, il C è un linguaggio sostanzialmente stabile da circa cinquant'anni e per questo utilizzabile, senza grossi stravolgimenti, in qualsiasi classe di elaboratori. Proprio per questo è ancora uno dei linguaggi più diffusi.

In conclusione, posso senz'altro affermare che questo è un testo di grande valore sia dal punto di vista tecnico che da quello didattico, la cui impostazione è maturata attraverso otto edizioni precedenti. Esso comprende argomenti molto preziosi e fondamentali per ogni informatico, quali il linguaggio C e la programmazione strutturata. Questi fanno parte dal punto di vista storico di quella rivoluzione informatica degli anni Settanta, vissuta professionalmente da molti di noi. In quegli anni la tecnologia rendeva il computer oggetto pervasivo e accessibile a tutti, e dai sistemi chiusi e proprietari si passava ai sistemi aperti. Nessuno si era allora reso conto di quelli che sarebbero stati i risvolti tecnologici, sociali e umani della rivoluzione che oggi viviamo pienamente.

*Salvatore Gaglio
Dipartimento di Ingegneria
Università degli Studi di Palermo*

Prefazione

Un innovativo manuale di programmazione in linguaggio C per gli anni 2020

“I bravi programmati scrivono codice che gli esseri umani possono capire.”¹

—Martin Fowler

“Penso che sia estremamente importante per noi, che ci occupiamo di informatica, continuare a divertirci nel nostro lavoro.”²

—Alan Perlis

Benvienuti nel C, uno dei linguaggi di programmazione più diffusi al mondo. Oltre a presentare un'introduzione al C semplice, attuale, ricca di codice e di casi pratici,³ questo libro ha molto da offrirvi, che siate studenti, docenti o programmati professionisti. In questa Prefazione vi presentiamo “l'anima del libro”.

Al centro del libro si colloca l'**approccio “live-code”** (letteralmente “codice dal vivo”) proprio del marchio Deitel. I concetti vengono presentati nel contesto di **147 programmi in C completi, funzionanti e relativi al mondo reale**, anziché con semplici frammenti di codice. Ogni esempio di codice è seguito da una o più finestre di input/output. Tutto il codice sorgente è disponibile online nella piattaforma MyLab.

Vi suggeriamo di eseguire ciascun programma contemporaneamente alla lettura del testo, in modo da ottimizzare il vostro processo di apprendimento.

Nel corso degli ultimi decenni:

- l'hardware dei computer è diventato rapidamente sempre più veloce, economico e di dimensioni ridotte;
- la larghezza di banda di Internet (cioè la capacità di trasportare informazioni) è aumentata a passo sostenuto, diventando meno costosa;
- è aumentata la possibilità di usufruire di software di qualità, spesso gratuitamente, o quasi, grazie al movimento **“open source”**.

Tratteremo diffusamente queste importanti tendenze. L'**“Internet delle cose”**(IoT, *Internet of Things*) connette già oggi decine di miliardi di dispositivi di ogni tipo immaginabile, che generano quantità enormi di dati (una forma di **“big data”**) a un ritmo in rapido aumento per velocità e volume. Si arriverà a svolgere la maggior parte delle attività informatiche online nel **“cloud”**, ovvero utilizzando servizi accessibili da Internet.

I primi capitoli del libro forniscono ai principianti basi solide nei fondamenti della programmazione. I capitoli intermedi e avanzati e gli oltre 20 casi pratici introducono i programmati principianti alle pratiche e alle sfide del mondo dello sviluppo software professionale.

Poiché le attuali applicazioni richiedono prestazioni eccezionali relativamente a hardware, software e Internet, spesso i professionisti scelgono il C per costruire le porzioni più ad alta intensità di prestazioni di queste applicazioni. Nel corso del libro ci concentreremo sugli aspetti relativi alle prestazioni per prepararvi meglio ad affrontare le problematiche a livello industriale.

L'architettura modulare del libro lo rende adatto per diversi tipi di esigenze.

1. Martin Fowler (con il contributo di Kent Beck). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999, p. 15.

2. Alan Perlis, citato nella dedica di *The Structure and Interpretation of Computer Programs*, 2/e di Hal Abelson, Gerald Jay Sussman e Julie Sussman. McGraw-Hill, 1996.

3. Tiobe Index del novembre 2020. Accesso 9 novembre 2020. <https://www.tiobe.com/tiobe-index/>.

- **Corsi universitari di programmazione introduttivi e intermedi** per studenti di informatica, ingegneria informatica, sistemi informativi, tecnologie dell'informazione, ingegneria del software e ambiti disciplinari collegati.
- **Corsi universitari di scienza, tecnologia, ingegneria e matematica (STEM)** che includono una parte relativa alla programmazione.
- **Corsi di aggiornamento aziendali e corsi professionalizzanti.**
- **Professionisti esperti** che apprendono il C in preparazione a nuovi progetti di sviluppo software.

Scrivere nove edizioni di questo libro negli ultimi 29 anni è stato un piacere. Speriamo che quest'ultima edizione sia per voi ricca di informazioni, stimolante e divertente mentre vi preparate a sviluppare applicazioni e sistemi avanzati e all'avanguardia importanti per la vostra carriera.

Caratteristiche nuove e aggiornate nella nona edizione

Elenchiamo qui di seguito brevemente alcune caratteristiche nuove e aggiornate di questa edizione. Ce ne sono molte altre e nei paragrafi successivi verranno forniti ulteriori dettagli.

- Alcuni casi pratici consistono in paragrafi nei quali viene spiegato tutto il codice sorgente, integrati da esercizi di fine capitolo nei quali vi viene richiesto di modificare il codice presentato nel testo o di affrontare compiti correlati. Altri casi sono esercizi con specifiche dettagliate tramite le quali dovreste essere in grado di sviluppare autonomamente il codice che rappresenta la soluzione. In altri casi si tratta di esercizi che vi rimandano a siti web con semplici tutorial. Poi ci sono esercizi in cui siete invitati a consultare siti web di sviluppatori dove potrete trovare codice da studiare ma senza tutorial, e il codice potrebbe non essere corredata da commenti esaustivi. Saranno i docenti a decidere quali casi pratici sono adatti a seconda delle esigenze.
- Aderiamo agli **standard C11/C18**.
- Abbiamo testato la correttezza del codice sui sistemi operativi **Windows, macOS e Linux**, utilizzando la versione più aggiornata dei **compilatori** rispettivamente di **Visual C++, Xcode e GNU gcc**, segnalando le differenze tra le piattaforme. Consultate la sezione “**Prima di cominciare**”, che segue la Prefazione, per le istruzioni relative all’installazione del software.
- Abbiamo usato lo **strumento di analisi statica del codice clang-tidy** per controllare il codice negli **esempi di codice** del libro e suggerire miglioramenti, che vanno da una semplice **verifica che le variabili siano inizializzate alla segnalazione di potenziali rischi di sicurezza**. Questo strumento è stato eseguito anche sul codice delle soluzioni di centinaia di esercizi del libro, rese disponibili per i docenti. L’elenco completo dei controlli è consultabile su <https://clang.llvm.org/extrclang-tidy/checks/list.html>.
- **GNU gcc** è tendenzialmente il compilatore più conforme al linguaggio C. Per consentire agli utenti **macOS e Windows** di utilizzare **gcc** se lo desiderano, il **Capitolo 1** include un test guidato che illustra come usare **gcc** per compilare ed eseguire programmi nel contenitore Docker della **GNU Compiler Collection multipiattaforma**.
- Abbiamo aggiunto più di **350 esercizi di autovalutazione**, con le relative risposte, ideali per l’apprendimento in autonomia e per l’uso nelle cosiddette “**flipped classroom**” o “**classi capovolte**” (vedi Paragrafo “**Flipped Classroom**” in questa Prefazione).
- Per garantire l’attualità del contenuto del libro, abbiamo svolto in Internet ricerche approfondite, in particolare sul C e in generale sul mondo dell’informatica, che hanno influito sulla nostra scelta di casi pratici. Mostriamo il C per l’uso a cui è destinato, con una vasta raccolta di casi pratici per la programmazione di applicazioni e di sistemi, focalizzando l’attenzione su **informatica, intelligenza artificiale, data science** e altri campi. Per approfondimenti, consultate il paragrafo “**Una panoramica dei casi pratici**” in questa Prefazione.
- Con il testo, gli esempi di codice, gli esercizi e i casi pratici aiutiamo gli studenti a familiarizzare con gli **attuali temi di interesse per gli sviluppatori**, tra cui: software open source, virtualizzazione, simulazione, servizi web, multithreading, architettura hardware multicore, programmazione di sistemi,

programmazione di giochi, animazione, visualizzazione, grafica 2D e 3D, intelligenza artificiale, elaborazione del linguaggio naturale, machine learning, robotica, data science, programmazione sicura, crittografia, Docker, GitHub, StackOverflow, forum e molto altro.

- Ci atteniamo alle più recenti **raccomandazioni di ACM/IEEE per i corsi di laurea in informatica**, che prevedono la trattazione dei concetti di sicurezza, data science, etica, privacy e prestazioni, oltre che l'utilizzo di dati del mondo reale durante tutto il programma di studi.
- Nelle edizioni più recenti di questo libro, la maggior parte dei capitoli si conclude con un paragrafo **“Programmazione sicura in C”** che si concentra sul *SEI CERT C Coding Standard* del Software Engineering Institute (SEI) della Carnegie Mellon University. In questa edizione abbiamo messo a punto le parti basate sul SEI CERT. Abbiamo anche aggiunto un'icona “sicurezza” a margine della pagina nei punti del testo nei quali si trattano temi relativi alla sicurezza. Tutto ciò è conforme al **maggior rilievo posto sulla sicurezza nella documentazione di ACM/IEEE per i corsi di laurea in informatica**.
- In linea con questo approfondimento sulla sicurezza, abbiamo aggiunto casi pratici su sistemi crittografici a chiave segreta e a chiave pubblica. Tra questi ultimi vi è una dettagliata spiegazione passo per passo del famosissimo algoritmo RSA, con suggerimenti utili a crearne una semplice implementazione su piccola scala.
- Abbiamo arricchito i casi pratici esistenti e ne abbiamo aggiunti altri dedicati a intelligenza artificiale (IA) e data science, che includono simulazioni con generazione di numeri casuali, analisi dei dati di sondaggi, elaborazione del linguaggio naturale (NLP, *Natural Language Processing*) e intelligenza artificiale (machine learning con regressione lineare semplice). La data science è messa in evidenza nei documenti più recenti di ACM/IEEE per i corsi di laurea in informatica.
- Abbiamo inoltre aggiunto esercizi nei quali gli studenti devono effettuare ricerche in Internet sulle problematiche di **etica e privacy** nell’informatica.
- Abbiamo perfezionato il nostro caso pratico sulle prestazioni multithreading e multicore. Abbiamo inoltre inserito un'icona “**prestazioni**” a margine della pagina nei punti nei quali si trattano temi relativi alle prestazioni.
- Per una lettura più scorrevole, abbiamo integrato direttamente nel testo le centinaia di suggerimenti per lo sviluppo del software presenti nelle edizioni precedenti. Gli **errori comuni e le buone pratiche di ingegneria del software** vengono segnalate con nuove icone a margine.
- Abbiamo aggiornato la nostra appendice riguardante ulteriori algoritmi di ricerca e analisi di algoritmi con “O grande” sviluppandola in un intero capitolo (Capitolo 13).
- I programmatore che usano il C spesso apprendono poi uno o più linguaggi orientati agli oggetti basati sul C. Abbiamo quindi inserito un’appendice contenente una semplice introduzione ai concetti e alla terminologia della programmazione orientata agli oggetti. Il C è un linguaggio di programmazione procedurale, quindi questa appendice aiuterà gli studenti a comprendere le differenze nell’approccio mentale tra i programmatore in C e chi invece utilizza linguaggi di programmazione come C++, Java, C#, Objective-C, Swift e altri linguaggi orientati agli oggetti. Abbiamo inserito molto altro di questo genere nel libro per preparare gli studenti ad affrontare il mondo dell’industria.
- In molti casi pratici utilizzerete ora librerie e strumenti gratuiti open source.
- Abbiamo aggiunto un caso pratico che esegue la visualizzazione con gnuplot.
- Abbiamo eliminato l’introduzione alla programmazione in C++ presente nell’edizione precedente per dare spazio alle centinaia di esercizi di autovalutazione e ai nostri nuovi casi pratici di programmazione di applicazioni e di sistemi.

Una panoramica del libro

I docenti possono facilmente adattare il contenuto per vari tipi di corsi e di destinatari. Presentiamo qui di seguito una breve panoramica del libro capitolo per capitolo, indicando la collocazione dei casi pratici. Alcuni sono esempi contenuti all’interno del capitolo, mentre altri sono esercizi di fine capitolo. Per alcuni troverete il codice completo, mentre in altri casi dovrete sviluppare la soluzione.

Nei Capitoli 1-5 vengono trattati i classici argomenti introduttivi alla programmazione in C. Nei Capitoli 6-11 troviamo gli argomenti intermedi, che costituiscono il livello alto dei corsi introduttivi di informatica e altri corsi correlati. I Capitoli 12-15 riguardano gli argomenti più avanzati, adatti ai corsi di informatica successivi. Ecco un elenco dei casi pratici, attuali, stimolanti e spesso divertenti.

Casi pratici per la programmazione di sistemi

- **Software dei sistemi:** costruire il proprio computer (come macchina virtuale).
- **Software dei sistemi:** costruire il proprio compilatore.
- **Programmazione di sistemi *embedded* (integriti):** robotica, grafica 3D e animazione con il simulatore Webots.
- **Prestazioni con sistemi multithreading e multicore.**

Casi pratici per la programmazione di applicazioni

- **Sviluppo di algoritmi:** iterazione controllata da contatore.
- **Sviluppo di algoritmi:** iterazione controllata da sentinella.
- **Sviluppo di algoritmi:** istruzioni di controllo annidate.
- **Simulazione di numeri casuali:** creare un gioco da casinò.
- **Simulazione di numeri casuali:** mescolare e distribuire carte da gioco.
- **Simulazione di numeri casuali:** la gara tra la tartaruga e la lepre.
- **Introduzione alla data science:** analisi dei dati di un sondaggio.
- **Elaborazione di file ad accesso diretto:** costruire un sistema per l'elaborazione di transazioni.
- **Visualizzazione di algoritmi di ricerca e di ordinamento:** ricerca binaria e ordinamento per fusione.
- **Intelligenza artificiale e data science:** elaborazione del linguaggio naturale (“Chi ha scritto davvero le opere di William Shakespeare?”)
- **Intelligenza artificiale e data science:** machine learning con la Libreria Scientifica GNU (“Le statistiche possono trarre in inganno” e “Le temperature medie di gennaio a New York City sono aumentate nel corso dell'ultimo secolo?”)
- **Programmazione di giochi:** gioco Cannon con la libreria raylib.
- **Programmazione di giochi:** gioco SpotOn con la libreria raylib.
- **Multimedia (audio e animazione):** la gara tra la tartaruga e la lepre con la libreria raylib.
- **Sicurezza e crittografia:** implementare un cifrario di Vigenère a chiave segreta e una crittografia RSA a chiave pubblica.
- **Visualizzazione animata con raylib:** la legge dei grandi numeri.
- **Servizi web e cloud:** ottenere un bollettino meteorologico utilizzando libcurl e i servizi web OpenWeatherMap; introduzione alla creazione di mashup con i servizi web.

La seguente descrizione riassuntiva dei capitoli del libro vi aiuterà a fare le scelte più adeguate, sia che siate studenti interessati ad avere un'idea del manuale che utilizzerete, docenti che pianificano il programma del corso o sviluppatori di software professionisti che decidono quali capitoli leggere per prepararsi a un progetto.

Parte 1: fondamenti di programmazione

Capitolo 1, “Introduzione ai computer e al linguaggio C”. Intende attrarre gli studenti principianti con dati statistici interessanti allo scopo di appassionarli allo studio e alla programmazione del computer. Il capitolo include una descrizione delle attuali tendenze della tecnologia, concetti di hardware e software e un'illustrazione della gerarchia dei dati, dai bit ai database. Pone le basi per le discussioni sulla programmazione in C che seguiranno nei Capitoli 2-15, nelle appendici e nei casi pratici integrati.

Il capitolo illustra i tipi di linguaggio di programmazione e le differenti tecnologie che probabilmente utilizzerete nello sviluppo di software. Introduciamo la Libreria Standard del C, con funzioni esistenti, riutilizzabili,

di alta qualità e con prestazioni elevate, che vi eviteranno di “reinventare la ruota”. Introduciamo inoltre **Internet**, il **World Wide Web**, il “**cloud**” e l’**Internet delle cose (IoT)**, che costituiscono la base per lo sviluppo delle applicazioni moderne.

I **test guidati** di questo capitolo mostrano come compilare ed eseguire codice in C con:

- **Visual C++ di Microsoft** in Visual Studio su Windows;
- **Xcode di Apple** su macOS;
- **gcc di GNU** su Linux.

Abbiamo eseguito i 147 esempi di codice del libro in ciascun ambiente.⁴ Potete scegliere l’ambiente di sviluppo software che preferite, in quanto il libro funziona bene anche con altri.

Mostriamo inoltre **GNU gcc nel contenitore Docker della GNU Compiler Collection**. Questo vi permette di eseguire il più recente compilatore **GNU gcc** su Windows, macOS o Linux; ciò è molto importante perché solitamente i compilatori GNU implementano tutte, o quasi, le funzionalità negli standard di linguaggio più recenti. Per le istruzioni di installazione del compilatore, consultate la sezione “Prima di cominciare”, che segue la Prefazione. Nel paragrafo “**Docker**” di questa Prefazione troverete ulteriori informazioni su questo strumento molto importante per gli sviluppatori. Per gli utenti Windows, indichiamo le istruzioni di Microsoft che vi permettono di installare Linux in Windows tramite il **sottosistema Windows per Linux (WSL, Windows Subsystem for Linux)**. Anche questo è un modo per poter usare il compilatore **GNU gcc** su Windows.

Apprenderete quanto grandi siano i “**big data**” e come stiano rapidamente crescendo. Il capitolo si conclude con un’introduzione all’**intelligenza artificiale (IA)**, una correlazione fondamentale tra i campi dell’informatica e della data science. Quasi sicuramente IA e data science svolgeranno un ruolo significativo nella vostra carriera informatica.

Capitolo 2, “Introduzione alla programmazione nel linguaggio C”. Presenta la basi del C e illustra le funzionalità fondamentali del linguaggio, inclusi input, output, principali tipi di dati, concetti relativi alla memoria, operatori aritmetici e regole di precedenza, processi decisionali.

Capitolo 3, “Sviluppo di un programma strutturato”. È uno dei capitoli più importanti per i programmati principianti. Si focalizza su **risoluzione di problemi e sviluppo di algoritmi con istruzioni di controllo** del C. Imparerete a **sviluppare algoritmi attraverso un processo top-down di affinamenti successivi**, usando le istruzioni di selezione **if e if...else**, l’istruzione di iterazione **while** per iterazione controllata da contatore e iterazione controllata da sentinella, e gli operatori di incremento, di decremento e di assegnazione. Il capitolo presenta tre casi pratici con sviluppo di algoritmi.

Capitolo 4, “Controllo nei programmi”. Presenta le ulteriori **istruzioni di controllo** in C, ossia **for, do...while, switch, break e continue**, e gli operatori logici. Un elemento essenziale di questo capitolo consiste nel **riepilogo dei principi della programmazione strutturata**.

Capitolo 5, “Funzioni”. Introduce la modalità di costruzione di programmi che utilizza funzioni esistenti e personalizzate come elementi base. Mostriamo **tecniche di simulazione con generazione di numeri casuali**. Descriviamo inoltre il passaggio delle informazioni tra le funzioni e come il meccanismo di chiamata/ritorno delle funzioni è supportato dalla pila delle chiamate delle funzioni e dai record di attivazione. Viene introdotto il concetto di ricorsione. Questo capitolo include anche il nostro primo caso pratico sulla simulazione, “**Creare un gioco da casinò**”, arricchito da esercizi a fine capitolo.

Parte 2: array, puntatori e stringhe

Capitolo 6, “Array”. Presenta la **struttura di dati array**, integrata in C, per rappresentare liste e tabelle di valori. Imparerete a definire e inizializzare un array e a fare riferimento ai singoli elementi che lo compongono. Discuteremo come passare gli array alle funzioni, ordinare ed effettuare ricerche negli array, manipolare array multidimensionali e creare array di lunghezza variabile la cui dimensione è determinata al momento dell’esecuzione. Nel capitolo vi è inoltre il nostro primo caso pratico sulla data science: “**Introduzione alla data science: analisi dei dati di un sondaggio**”. Negli esercizi si trovano due casi pratici sulla **programmazione di**

4. I pochi casi in cui il compilatore non supporta una particolare funzionalità vengono segnalati esplicitamente.

giochi con grafica, suoni e rilevamento delle collisioni e un caso pratico sulla **programmazione di sistemi embedded (robotica con il simulatore Webots)**.

Capitolo 7, “Puntatori”. Presenta la funzionalità che si può considerare la più potente del C. I puntatori permettono ai programmi di:

- realizzare il passaggio per riferimento;
- passare le funzioni ad altre funzioni;
- creare e manipolare strutture dinamiche di dati, che studierete in dettaglio nel **Capitolo 12**.

Il capitolo spiega i concetti fondamentali sui puntatori, quali la dichiarazione e l'inizializzazione dei puntatori, come ottenere l'indirizzo in memoria di una variabile, la dereferenziazione dei puntatori, l'aritmetica dei puntatori e la stretta correlazione tra array e puntatori. Questo capitolo presenta il nostro primo caso pratico sul software dei sistemi, **“Costruite il vostro computer come macchina virtuale”**, con l'utilizzo della simulazione, che introduce un elemento essenziale della moderna architettura del computer, la **macchina virtuale**.

Capitolo 8, “Caratteri e stringhe”. Introduce le funzioni della Libreria Standard del C per l'elaborazione di stringhe, caratteri e blocchi di memoria. Utilizzerete queste potenti funzionalità nel **Capitolo 11, “Elaborazione di file”**, affrontando un caso pratico sull'**elaborazione del linguaggio naturale (NLP)**. Vedrete anche la stretta relazione esistente tra puntatori e array.

Parte 3: input/output formattato, strutture ed elaborazione di file

Capitolo 9, “Input/output formattato”. Tratta le potenti funzionalità di formattazione delle funzioni `scanf` e `printf`. Se utilizzate correttamente, queste funzioni **in sicurezza** rispettivamente leggono dati dallo stream (letteralmente “flusso”) standard `input` e inviano in uscita dati allo stream standard `output`.

Capitolo 10, “Strutture, unioni, manipolazione di bit ed enumerazioni”. Introduce: strutture (`struct`) per aggregare insiemi di dati correlati in tipi personalizzati; unioni per la condivisione della memoria tra diverse variabili; `typedef` per creare alias per tipi di dati definiti in precedenza; operatori bit a bit per manipolare i singoli bit degli operandi interi; enumerazioni per definire insiemi di costanti intere denominate. Molti programmati in C proseguono studiando C++ e la programmazione orientata agli oggetti. Nel linguaggio C++, le strutture del C si evolvono in classi, che sono i “modelli” che i programmati in C++ usano per creare oggetti. Le `struct` di C contengono solo dati. Le classi di C++ possono invece contenere dati e funzioni.

Capitolo 11, “Elaborazione di file”. Introduce file per la memorizzazione a lungo termine dei dati, anche quando il computer è spento. I dati di questo tipo vengono detti “persistenti”. Il capitolo spiega come vengono creati, aggiornati ed elaborati i file di testo e i file binari. Analizziamo l'elaborazione sia dei file ad accesso sequenziale che dei file ad accesso casuale. In uno degli esercizi leggerete i dati da un file CSV (*Comma-Separated Values*), ossia con valori separati da virgolette. Il formato CSV è uno tra i più diffusi nella community della data science. Questo capitolo presenta il nostro caso pratico successivo, **la creazione di un sistema per l'elaborazione di transazioni usando file ad accesso casuale**. Utilizziamo file ad accesso casuale per simulare il tipo di capacità di accesso diretto ad alta velocità che hanno i sistemi di gestione di database di livello industriale. Sempre in questo capitolo troviamo anche il nostro primo caso pratico su intelligenza artificiale/data science, che usa le tecniche di **elaborazione del linguaggio naturale (NLP)** per iniziare un'indagine sulla controversa questione: “Chi ha scritto davvero le opere di William Shakespeare?”. Un secondo caso pratico su intelligenza artificiale/data science, **“Machine learning con la Libreria Scientifica GNU”**, usa la regressione lineare semplice per esaminare il quartetto di Anscombe.⁵ Esso è composto da quattro dataset totalmente diversi tra loro, che hanno statistiche descrittive di base praticamente identiche. Offre spunti preziosi a studenti e sviluppatori che stanno apprendendo i concetti base della data science in questo manuale di informatica. Il caso pratico vi chiede quindi di eseguire una regressione lineare semplice sui dati relativi alle temperature medie di gennaio a New York City nel corso di 126 anni, per verificare se la tendenza è verso il raffreddamento oppure il riscaldamento.

⁵ “Anscombe's Quartet.” Accesso 13 novembre 2020. https://en.wikipedia.org/wiki/Anscombe%27s_quartet.

5. “Anscombe's Quartet.” Accesso 13 novembre 2020. https://en.wikipedia.org/wiki/Anscombe%27s_quartet.

Parte 4: algoritmi e strutture di dati

Capitolo 12, “Strutture di dati”. Discute l’uso delle strutture (`struct`) per aggregare insiemi di dati correlati in tipi personalizzati, l’uso di `typedef` per creare alias per tipi di dati definiti in precedenza, e strutture di dati collegate in modo dinamico che possono crescere e ridursi in fase di esecuzione: liste collegate, pile, code e alberi binari. Potete usare le tecniche apprese per implementare altre strutture di dati. Questo capitolo presenta, negli esercizi, il nostro successivo caso pratico sul software dei sistemi, “Costruire il proprio compilatore”. Definiremo un semplice ma potente linguaggio di alto livello. Scrivrete 00alcuni programmi con linguaggio di alto livello, che il vostro compilatore tradurrà nel linguaggio macchina del computer che avrete costruito nel Capitolo 7. Il compilatore metterà l’output risultante in un file, quindi il vostro computer caricherà (`load`) in memoria il linguaggio macchina dal file, lo eseguirà e produrrà gli output appropriati.

Capitolo 13, “Pensare come un informatico: algoritmi di ordinamento e O grande”. Introduce alcuni temi classici di informatica. Prendiamo in esame numerosi algoritmi, mettendone a confronto le necessità relativamente al processore e al consumo di memoria. Presentiamo una chiara introduzione alla notazione **O grande**, che indica quanto sforzo può essere richiesto a un algoritmo per risolvere un problema, in base al numero di elementi che deve elaborare. Il capitolo include anche il caso pratico “Visualizzazione dell’algoritmo di ordinamento per fusione ad alte prestazioni”.

Per il programma di un corso sulle strutture di dati in C, si può fare riferimento ai seguenti contenuti: ricorsione (Capitolo 5), array (Capitolo 6), ricerca (Capitolo 6), strutture di dati (Capitolo 12), ordinamento (Capitolo 13) e **O grande** (Capitolo 13).

Parte 5: preprocessore e altri argomenti

Capitolo 14, “Preprocessore”. Discute ulteriori funzionalità del preprocessore del C, quali: usare `#include` per la gestione di file in programmi di grandi dimensioni; usare `#define` per creare macro con e senza argomenti; usare la compilazione condizionale per specificare porzioni di un programma che non sempre devono essere compilate (come il codice supplementare usato solo durante lo sviluppo del programma); stampare messaggi di errore durante la compilazione del programma; usare asserzioni per verificare se i valori di alcune espressioni sono corretti.

Capitolo 15, “Altri argomenti”. Copre ulteriori argomenti del C, inclusi: liste di argomenti di lunghezza variabile, argomenti della riga di comando, compilazione di programmi con più file sorgente, dichiarazioni `extern` per variabili globali in altri file, prototipi di funzioni, limitazione del campo d’azione con `static`, terminazione di programmi con `exit` e `atexit`, suffissi per letterali interi e in virgola mobile, gestione dei segnali, funzioni `calloc` e `realloc` per l’allocazione dinamica della memoria, salto non condizionato con `goto`.

Appendici

Appendice A, “Tabella di precedenza degli operatori”. Elenca gli operatori del C in ordine decrescente di precedenza.

Appendice B, “Insieme di caratteri ASCII”. Mostra i caratteri e il loro codice numerico corrispondente.

Appendice C, “Multithreading/multicore e altri argomenti di C18/C11/C99”. Tratta inizializzatori designati, letterali composti, tipo `bool`, numeri complessi, la parola chiave `restrict`, divisione intera affidabile, membri di array flessibili, tipo `math` generico, funzioni inline, identificatore predefinito `__func__`, macro `va_copy`, file di intestazione C11, parola chiave `_Generic` (espressioni di tipo generico), funzione `quick_exit`, supporto Unicode®, specificatore di funzione `_noreturn`, espressioni di tipo generico. Annex L e analizzabilità e comportamento non definito, controllo dell’allineamento della memoria, asserzioni statiche e tipi in virgola mobile. Questa appendice presenta il nostro caso pratico finale, “Prestazioni con sistemi multithreading e multicore”, che mostra come creare programmi multithread che verranno eseguiti più rapidamente (e spesso molto più rapidamente) sulle attuali architetture di computer multicore. Questo caso pratico è il coronamento adatto per un libro sul C, per il quale è di fondamentale importanza scrivere programmi ad alte prestazioni.

Appendice D, “Introduzione ai principi della programmazione orientata agli oggetti”. Introduce in modo semplice la terminologia e i concetti della programmazione orientata agli oggetti. Quando avrete appreso il C,

probabilmente studierete anche uno o più linguaggi di programmazione orientata agli oggetti basati sul C, quali C++, Java, C#, Objective-C o Swift, usandoli in affiancamento al C.

Appendici online

L'Appendice E, "Sistemi di numerazione", introduce i sistemi di numerazione binario, ottale, decimale ed esadecimale. Le Appendici F-H ("Utilizzo del debugger Visual Studio", "Utilizzo del debugger GNU", "Utilizzo del debugger di Xcode") mostrano come utilizzare le principali funzionalità di debugging dei nostri compilatori preferiti per individuare e correggere i problemi in fase di esecuzione nei programmi.

Caratteristiche della nona edizione

Fondamenti di programmazione in C

Nel libro offriamo un'ampia copertura dei **fondamenti della programmazione in C**:

- è stata data particolare importanza alla **risoluzione dei problemi** e allo **sviluppo di algoritmi**;
- per preparare gli studenti all'ambiente professionale, abbiamo utilizzato la terminologia della **documentazione C standard** più recente piuttosto che i termini generali della programmazione;
- **abbiamo evitato la trattazione dei concetti matematici più complessi**, lasciandola per corsi di livello elevato, includendo comunque **esercizi matematici facoltativi** per gli studenti dei corsi scientifici e di ingegneria.

C11 e C18 standard

Il C11 amplia e migliora le funzionalità del C. Abbiamo introdotto ulteriori funzionalità del C11 standard. Dopo il C11, c'è stata solo un'altra versione, il C18,⁶ che "ha risolto i problemi del C11 senza introdurre nuove funzionalità del linguaggio".⁷

Nuovo approccio pedagogico con più di 350 esercizi di autovalutazione

In questo libro usiamo il nostro nuovo approccio pedagogico, che prevede esercizi di autovalutazione con risposte, introdotto nel nostro recente manuale sulla programmazione in Python, *Introduzione a Python per l'informatica e la data science*.

- I paragrafi all'interno dei capitoli sono volutamente brevi, secondo il **metodo "leggere, fare, verificare"**. Dopo aver letto l'introduzione a un nuovo concetto, studierete ed eseguirete gli esempi di codice corrispondenti, quindi verificherete l'avvenuta comprensione tramite gli **esercizi di autovalutazione seguiti immediatamente dalle relative risposte**. Questo metodo di studio vi aiuterà a mantenere un buon ritmo di apprendimento.
- Le **tipologie di autovalutazione (completare, vero/falso, discussione)** vi permetteranno di verificare la vostra comprensione della terminologia e dei concetti studiati.
- Le **autovalutazioni basate sul codice** vi daranno la possibilità di usare la terminologia appresa e di rinforzare le tecniche di programmazione appena imparate.
- Le **autovalutazioni** sono particolarmente utili nei corsi con approccio didattico **flipped classroom**, un fenomeno diffuso di cui parleremo a breve.

Semplicità, brevità, attualità

- **Semplicità:** i nostri obiettivi sono la **semplicità e la chiarezza**.
- **Brevità:** la maggior parte degli esempi del libro sono brevi. Quando se ne presenta la necessità, usiamo esempi di codice, esercizi e progetti più consistenti, soprattutto nei casi pratici, che sono una caratteristica fondamentale di questo manuale.

6. ISO/IEC 9899:2018, Information technology — Programming languages — C, <https://www.iso.org/standard/74528.html>.

7. [https://en.wikipedia.org/wiki/C18_\(C_standard_revision\)](https://en.wikipedia.org/wiki/C18_(C_standard_revision)) e http://www.iso-9899.info/wiki/The_Standard.

- **Attualità:** “Chi osa insegnare non deve mai smettere di imparare.”⁸ (J. C. Dana). Nelle nostre ricerche, abbiamo consultato, letto o guardato migliaia di articoli attuali, pubblicazioni di ricercatori, libri bianchi, libri, video, webinar, post su blog e forum, documentazione varia e molto altro.

Centinaia di esempi, esercizi e progetti (EEP) attuali

Userete un **approccio pratico e concreto**, imparando da una vasta selezione di **esempi, esercizi e progetti (EEP)** tratti dai campi dell’informatica, della data science e altri ancora.

- Affronterete sfide emozionanti e avvincenti nei nostri casi pratici più ampi, come: creare un gioco da casinò, sviluppare un programma per l’analisi dei dati di un sondaggio, creare un sistema di elaborazione di transazioni, costruire il proprio computer (usando la simulazione per creare una **macchina virtuale**), utilizzare tecnologie **IA/data science** come **l’elaborazione del linguaggio naturale** e **il machine learning**, costruire il proprio compilatore, programmare giochi per computer, programmare **simulazioni di robot con Webots** e scrivere codice multithread che vi permette di sfruttare le moderne architetture di computer multicore e ottenere così la massima prestazione dalla vostra macchina.
- **Le ricerche e i progetti negli esercizi** vi incoraggeranno ad approfondire ciò che avrete imparato e a esplorare altre tecnologie. Vi esortiamo a usare computer e Internet per risolvere problemi significativi. I progetti spesso sono di portata molto più vasta degli esercizi e alcuni potrebbero richiedere giorni o anche settimane di lavoro di implementazione. Molti di questi sono adatti per **progetti e ricerche per corsi universitari e tesi**. Per i progetti non vengono fornite le soluzioni.
- I docenti possono adattare i propri corsi a seconda delle particolari esigenze dei partecipanti, con la possibilità di variare di volta in volta esercizi e domande per test ed esami.

Lavorare con software open source

A quei tempi i programmatore [che usavano l’elaborazione batch] non documentavano mai i loro programmi, perché si presumeva che nessun altro li avrebbe mai usati. Ora, invece, il time-sharing ha reso semplice lo scambio di software: ne avete appena memorizzata una copia nel repository pubblico e quindi di fatto l'avete data al mondo intero. Immediatamente le persone iniziarono a documentare i propri programmi e a considerarli come utilizzabili da altri. Cominciarono a costruire gli uni sul lavoro degli altri.⁹

— Robert Fano, Fondatore e Direttore, negli anni Sessanta del secolo scorso, del Project MAC del MIT, evolutosi nell’odierno Computer Science and Artificial Intelligence Laboratory (CSAIL).¹⁰

Il software open source è «un software con un codice sorgente che chiunque può ispezionare, modificare e migliorare».¹¹ Vi incoraggiamo a guardare e provare le molte **demo** ed esempi di codice **open source** (disponibili su siti come **GitHub**) per trarne ispirazione. Parleremo più a fondo di GitHub nel paragrafo “Pensare come uno sviluppatore: GitHub, StackOverflow e altro”.

Visualizzazioni

Includiamo **visualizzazioni** di alto livello prodotte con il pacchetto di visualizzazione open source di **gnuplot** per consolidare la vostra comprensione dei concetti.

- Usiamo le **visualizzazioni** come strumento pedagogico. Per esempio, attraverso la **simulazione del lancio di dadi** “rendiamo viva” la **legge dei grandi numeri** (vedi più avanti nella Prefazione, “**Capitolo 10 - Casi pratici sulla programmazione di giochi con raylib**”). Mentre il programma aumenta il numero dei lanci di dado, vedrete le percentuali relative a ciascuna delle sei facce (1, 2, 3, 4, 5, 6)

8. John Cotton Dana. Da <https://www.bartleby.com/73/1799.html>: “Nel 1912 venne chiesto a Dana, un bibliotecario di Newark, in New Jersey, di trovare una citazione latina adatta per l’iscrizione su un nuovo edificio del Newark State College (ora Kean University), a Union, in New Jersey. Non riuscendo a trovare una citazione adeguata, Dana compose la frase che divenne il motto del college. *The New York Times Book Review*, 5 marzo 1967, p. 55”.

9. Robert Fano, citato in *Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal* di Mitchell Waldrop. Penguin Putnam, 2002, p. 232.

10. “MIT Computer Science and Artificial Intelligence Laboratory.” Accesso 9 novembre 2020. https://en.wikipedia.org/wiki/MIT_Computer_Science_and_Artificial_Intelligence_Laboratory.

11. “What is open source?” Accesso 14 novembre 2020. <https://opensource.com/resources/what-open-source>.

avvicinarsi gradualmente a 16,667% (1/6), mentre le altezze delle barre che rappresentano le percentuali si equalizzeranno.

- Vi suggeriamo di sperimentare con il codice per creare le vostre visualizzazioni.

Esperienze con i dati

Negli esempi, esercizi e progetti del libro (soprattutto nel capitolo sull’elaborazione dei file) lavorerete con **dati del mondo reale**, come per esempio l’opera di Shakespeare *Romeo and Juliet*. Scaricherete e analizzerete file di testo dal Progetto Gutenberg, una grossa fonte di testi da analizzare scaricabili gratuitamente. Il sito contiene circa 63.000 e-book in vari formati, inclusi file di testo puri, che non sono più coperti da copyright negli Stati Uniti. Lavorerete inoltre con dati relativi a temperature del mondo reale. In particolare, analizzerete 126 anni di dati relativi alle temperature medie di gennaio a New York City per determinare se la tendenza è verso il riscaldamento oppure il raffreddamento. Ricaverete questi dati dal sito web del National Oceanic and Atmospheric Administration (NOAA), noaa.gov.

Pensare come uno sviluppatore: GitHub, StackOverflow e altro

Il modo migliore per prepararsi [a essere un programmatore] è quello di scrivere programmi, e di studiare gli ottimi programmi scritti da altri. Nel mio caso, ho cercato nei cestini della spazzatura al Computer Science Center e ripescato i listati dei loro sistemi operativi.¹²

—William Gates

- In preparazione alla vostra carriera, lavorerete con siti molto usati dagli sviluppatori, come **GitHub** e **StackOverflow**, e farete molte ricerche su Internet.
- **StackOverflow** è uno dei più diffusi siti di domande e risposte pensati per gli sviluppatori.
- La community open source del C è molto vasta. Per esempio, su **GitHub** ci sono 32.000¹³ repository di codice in C! Su GitHub potete trovare codice in C sviluppato da altri e, se volete, potete anche costruire su di esso. Questo è un ottimo modo per imparare ed è una naturale estensione della nostra filosofia di insegnamento “live-code”, basata su un approccio diretto al codice.¹⁴
- **GitHub** è un luogo ideale per **trovare codice sorgente gratuito open source** da incorporare nei vostri progetti, e per contribuire con il vostro codice, se lo desiderate, alla **community open source**. Cinquanta milioni di sviluppatori usano GitHub.¹⁵ Il sito attualmente ospita oltre 100 milioni di repository per codice scritti in svariati linguaggi;¹⁶ solo nel corso del 2019, gli sviluppatori hanno contribuito con più di 44 milioni di repository.¹⁷ **GitHub** è un elemento cruciale nell’arsenale di uno sviluppatore software grazie agli **strumenti di controllo della versione** che aiutano i team di sviluppatori a gestire progetti pubblici open source e progetti privati.
- Nel 2018, Microsoft ha acquistato **GitHub** per 7,5 miliardi di dollari. Se diventerete sviluppatori software, è quasi certo che userete GitHub regolarmente. Secondo il CEO di Microsoft, Satya Nadella, l’acquisizione è stata fatta per «consentire a ogni sviluppatore di creare, innovare e risolvere le più urgenti sfide mondiali».¹⁸
- Vi esortiamo a studiare ed eseguire una grande quantità di codice open source sviluppato in C su GitHub.

Privacy

Nelle raccomandazioni di ACM/IEEE (le due principali associazioni di informatici e ingegneri) per i corsi di laurea in informatica, tecnologie dell’informazione e sicurezza informatica, viene menzionata la parola “pri-

12. William Gates, citato in *Programmers at Work: Interviews With 19 Programmers Who Shaped the Computer Industry* di Susan Lammers. Microsoft Press, 1986, p. 83; <https://www.microsoft.com/en-us/old-site/1986/interviews/gates.htm>.

13. “C.” Accesso 4 gennaio 2021. <https://github.com/topics/c>.

14. È necessario che gli studenti familiarizzino con l’ampia gamma di licenze open source per software su GitHub.

15. “GitHub.” Accesso 14 novembre 2020. <https://github.com/>.

16. “GitHub is how people build software.” Accesso 14 novembre 2020. <https://github.com/about>.

17. “The State of the Octoverse.” Accesso 14 novembre 2020. <https://octoverse.github.com>.

18. “Microsoft to acquire GitHub for \$7.5 billion.” Accesso 14 novembre 2020. <https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/>.

vacy” più di 200 volte. Chiunque studi e lavori nel campo della programmazione deve essere profondamente consapevole delle questioni e delle problematiche relative alla privacy. Gli studenti faranno ricerche sulla privacy in quattro esercizi nei Capitoli 1, 3 e 10.

Negli esercizi del Capitolo 1 inizierete a considerare questi temi effettuando ricerche in merito alle normative sempre più rigorose sulla privacy: per quanto riguarda gli Stati Uniti, la legge sulla portabilità e responsabilità delle polizze di assicurazione sanitaria (**HIPAA, Health Insurance Portability and Accountability Act**) e la legge sulla privacy dei consumatori in California (**CCPA, California Consumer Privacy Act**); per quanto riguarda l’Unione Europea, il regolamento generale sulla protezione dei dati (**GDPR, General Data Protection Regulation**).

Etica

Nelle raccomandazioni di ACM per i corsi di laurea in informatica, tecnologie dell’informazione e sicurezza informatica viene menzionata la parola “etica” più di 100 volte. In molti esercizi del Capitolo 1 vi focalizzerete su questioni etiche facendo ricerche in Internet. Approfondirete problematiche etiche e di privacy relative agli assistenti intelligenti, come **Watson di IBM**, **Alexa di Amazon**, **Siri di Apple**, l’**Assistente Google** e **Cortana di Microsoft**. Per esempio, è accaduto che un giudice in New Hampshire ordinasse ad Amazon di consegnare le registrazioni di Alexa per utilizzarle in un procedimento penale.¹⁹

Prestazioni

I programmati preferiscono il linguaggio C (e il C++) per sistemi caratterizzati da prestazioni intensive come sistemi operativi, sistemi in tempo reale, sistemi embedded (incorporati), sistemi di gioco e sistemi di comunicazione. Per questo motivo **ci concentriamo sugli aspetti relativi alle prestazioni**. Usiamo **operazioni di cronometraggio** nei nostri esempi di multithreading per misurare l’incremento delle prestazioni che possiamo ottenere sui sistemi multicore più popolari aumentando il numero di core.

Strumenti di analisi statica del codice

Gli strumenti di analisi statica del codice vi permettono di controllare rapidamente che nel vostro codice non vi siano errori e problemi di sicurezza, fornendovi anche spunti per migliorarlo. Abbiamo controllato tutto il nostro codice in C usando lo strumento **clang-tidy** (<https://clang.llvm.org/extra/clang-tidy/>). Abbiamo inoltre utilizzato il flag del compilatore **-Wall** nei compilatori GNU gcc e Clang per abilitare tutti i messaggi di avvertimento del compilatore. Salvo alcune eccezioni per i casi che esulano dall’ambito di questo libro, ci assicuriamo che la compilazione dei nostri programmi avvenga senza messaggi di avvertimento.

Gestione dell’Annex K del C11 e di printf_s/scanf_s

L’Annex K del C11 standard introduce versioni più sicure di **printf** (per l’output) e **scanf** (per l’input) chiamate **printf_s** e **scanf_s**. Analizzeremo queste funzioni e le questioni di sicurezza corrispondenti nei Paragrafi 6.13 e 7.13.

- L’Annex K è opzionale, per cui non sarà implementato da tutti i fornitori del linguaggio C. In particolare, GNU C++ e Clang C++ non implementano l’Annex K, quindi l’utilizzo di **scanf_s** e **printf_s** potrebbe compromettere la portabilità del vostro codice per gli altri compilatori.
- Microsoft ha implementato le sue versioni Visual C++ di **printf_s** e **scanf_s** prima della pubblicazione del C11 standard. Il suo compilatore ha cominciato immediatamente a emettere messaggi di avvertimento per ogni chiamata di **scanf**, avvisando che **scanf** è stata *deprecata* (non deve essere più utilizzata) e che deve essere preso in considerazione invece l’uso di **scanf_s**. Ora Microsoft gestisce diversamente l’uso di **scanf**, che non genera più messaggi di avvertimento ma viene considerato come errore. Per impostazione predefinita, in Visual C++ un programma con **scanf** non verrà compilato. Il test guidato di Visual C++ nel Capitolo 1 mostra come gestire questo problema e compilare i nostri programmi.

19. “Judge orders Amazon to turn over Echo recordings in double murder case.” Accesso 14 novembre 2020. <https://techcrunch.com/2018/11/14/amazon-echo-recordings-judge-murder-case/>.

- Molte organizzazioni hanno standard di codifica che richiedono che il codice venga compilato senza messaggi di avvertimento. Vi sono due modi per evitare che vengano emessi per `scanf` in Visual C++: potete usare `scanf_s` invece di `scanf`, oppure potete disattivarli.
- Si sta discutendo se rimuovere l'Annex K dal C standard. Per questo motivo, usiamo `printf`/`scanf` nel corso di tutto il libro e spieghiamo agli utenti di Visual C++ come disabilitare gli errori `printf`/`scanf` impostati da Microsoft. Gli utenti Windows che preferiscono evitare questa operazione possono usare il compilatore `gcc` nel contenitore Docker della GNU GCC, discusso nel Paragrafo "Docker" di questa Prefazione. Per ulteriori dettagli, consultate la sezione "Prima di cominciare", che segue la Prefazione, e il Paragrafo 1.10.

Nuova appendice: "Introduzione ai principi della programmazione orientata agli oggetti"

Il modello di programmazione del linguaggio C è chiamato **programmazione procedurale**. Viene da noi insegnato come **programmazione procedurale strutturata**. Dopo che avrete appreso il C, probabilmente studierete anche uno o più linguaggi di programmazione orientata agli oggetti basati sul C, come Java, C++, C#, Objective-C o Swift, usandoli in affiancamento al C. Molti di questi linguaggi supportano diversi paradigmi di programmazione tra programmazione procedurale, programmazione orientata agli oggetti, programmazione generica e programmazione funzionale. Nell'**Appendice D** presentiamo una sintesi dei principi fondamentali della programmazione orientata agli oggetti.

Una panoramica dei casi pratici

Insieme ad altri esempi, esercizi e progetti abbiamo incluso nei capitoli molti casi pratici, di più ampia portata, adatti a corsi di programmazione introduttivi. Spetterà ai docenti selezionare quelli più appropriati per i singoli corsi.

Capitolo 5 - Simulazione di numeri casuali: creare un gioco da casinò

In questo caso pratico, userete generazione di numeri casuali e tecniche di simulazione per implementare il gioco di dadi noto come "Craps", molto popolare nei casinò.

Capitolo 5 - Simulazione di numeri casuali: la gara tra la tartaruga e la lepre

In questo caso pratico negli esercizi userete generazione di numeri casuali e tecniche di simulazione per implementare la famosa gara tra la tartaruga e la lepre.

Capitolo 6 - Visualizzazione della ricerca binaria

In questo caso pratico apprenderete l'algoritmo di ricerca binaria ad alta velocità e vedrete con una visualizzazione come l'effetto di dimezzamento dell'algoritmo consenta di ottenere prestazioni elevate.

Capitolo 6 - Introduzione alla data science: analisi dei dati di un sondaggio

In questo caso pratico apprenderete diverse statistiche descrittive basilari (media, mediana e moda) che sono comunemente usate per "conoscere i dati". Creerete poi un'applicazione per la manipolazione di array in grado di calcolare queste statistiche relativamente a un gruppo di dati di un sondaggio.

Capitolo 7 - Simulazione di numeri casuali: mescolare e distribuire carte da gioco

In questo caso pratico, userete array di stringhe, generazione di numeri casuali e tecniche di simulazione per implementare un programma basato su testo per mescolare e distribuire carte da gioco.

Capitolo 7 - Programmazione di sistemi embedded: robotica con il simulatore Webots

Webots (<https://cyberbotics.com/>) è un eccezionale simulatore open source di robotica 3D, eseguibile su Windows, macOS e Linux. Include simulazioni per dozzine di robot che camminano, volano, rotolano, guidano e altro ancora:

<https://cyberbotics.com/doc/guide/robots>

Userete il livello gratuito del simulatore di robotica Webots per esplorare le dozzine di simulazioni di robot. Esegirete diverse simulazioni di robotica 3D a colori scritte in C, studiandone il codice fornito. Webots è

un ambiente di sviluppo autonomo che fornisce un editor di codice e un compilatore per il C. Userete questi strumenti per **programmare le vostre simulazioni** con i robot di Webots.

Webots fornisce una moltitudine di programmi con codice completo in C. Un ottimo metodo per imparare il linguaggio C è studiare i programmi esistenti, modificarli affinché lavorino in modo leggermente diverso e osservare i risultati. Molte importanti aziende di robotica si affidano ai simulatori Webots per lo sviluppo dei prototipi di nuovi prodotti.

Capitolo 7 - Caso pratico sul software dei sistemi: costruite il vostro computer come macchina virtuale (usando la simulazione)

Nel corso di una serie di esercizi potrete "aprire" un ipotetico computer e osservarne la struttura interna. Introdurremo una semplice **programmazione in linguaggio macchina** e scriveremo diversi brevi programmi per questo computer, che chiameremo **Simpletron**. Come indica il suo nome, si tratta di una macchina semplice, ma, come vedremo, allo stesso tempo potente. Il Simpletron esegue programmi scritti nell'unico linguaggio che comprende direttamente, vale a dire il **linguaggio macchina del Simpletron (SML, Simpletron Machine Language)**. Affinché questa risulti un'esperienza particolarmente utile, **costruirete un computer** (con la tecnica della **simulazione software**) sul quale potrete eseguire i vostri programmi in linguaggio macchina! L'esperienza con il Simpletron vi offrirà un'introduzione basilare alla nozione di **macchine virtuali**, uno dei più importanti concetti di architettura dei sistemi nell'informatica moderna.

Capitolo 8 - Pqyoaf X Nylfomigrob Qwbbfmh Mndogvk: Rboqlrut yua Boklnxhmywex

Il titolo di questo caso pratico negli esercizi sembra indecifrabile. Non si tratta di un errore! Infatti in questo esercizio introduciamo la **crittografia**, che è di importanza fondamentale nel nostro mondo moderno interconnesso. Ogni giorno, dietro le quinte, la crittografia viene usata per **assicurare la privacy e la sicurezza delle vostre comunicazioni via Internet**. Con questo esercizio insistiamo sul tema della sicurezza, facendovi studiare l'algoritmo del **cifrario a chiave segreta di Vigenère** per poi implementarlo usando le tecniche di elaborazione degli array.²⁰ Lo userete quindi per criptare e decriptare vari messaggi e per decriptare il titolo di questo paragrafo.

Capitolo 8 - Crittografia RSA a chiave pubblica

Crittografia e decrittografia a chiave segreta hanno un difetto: un messaggio criptato può essere decriptato da chiunque scopra o sottragga la chiave segreta. Esploriamo la crittografia a chiave pubblica con l'algoritmo RSA. Questa tecnica esegue la crittografia con una chiave pubblica nota a chiunque voglia inviare un messaggio segreto a un destinatario particolare. La chiave pubblica può essere utilizzata per criptare i messaggi ma non per decriptarli. I messaggi possono essere decriptati unicamente con un'abbinata chiave privata che solo il destinatario conosce, è quindi una tecnica molto più sicura della chiave segreta usata nella crittografia a chiave segreta. La RSA è tra le tecnologie di crittografia a chiave pubblica più utilizzate al mondo. Creerete voi stessi una versione funzionante, su piccola scala, del sistema crittografico RSA.

Capitolo 10 - Casi pratici sulla programmazione di giochi con raylib

Troverete una serie di esercizi (**5 casi pratici e 10 esercizi supplementari**) nei quali userete **raylib²¹**, una libreria open source multiplattforma per la programmazione di giochi, che supporta Windows, macOS, Linux e altre piattaforme. Il team di sviluppo di **raylib** fornisce numerose **demo in linguaggio C** per aiutarvi ad apprendere le funzionalità e le caratteristiche principali di questa libreria. Studierete due giochi con codice completo e una visualizzazione dinamica animata che abbiamo creato noi.

- Il gioco Spot-On mette alla prova i vostri riflessi, chiedendovi di fare clic su punti in movimento prima che spariscano. A ogni nuovo livello, la velocità di movimento dei punti aumenta, rendendo il gioco più difficile.
- Il gioco Cannon vi sfida a mirare e sparare ripetutamente con un cannone per distruggere nove obiettivi in movimento prima dello scadere del tempo a disposizione. Il gioco è reso più difficile da un blocco che si sposta continuamente.

20. "Vigenère Cipher." Accesso 22 novembre 2020. https://en.wikipedia.org/wiki/Vigenère_cipher.

21. "raylib." Accesso 14 novembre 2020. <https://www.raylib.com>.

- Nella visualizzazione dinamica animata della “legge dei grandi numeri” viene lanciato ripetutamente un dado a sei facce e creato un **grafico a barre animato**. Le **visualizzazioni** sono un potente mezzo per comprendere i dati, molto più efficace che non semplicemente guardare ai dati “grezzi” (raw). Questo esercizio consente agli studenti di vedere la “legge dei grandi numeri” all’opera. Quando lanciamo ripetutamente un dado, ci aspettiamo che ciascuna faccia appaia all’incirca 1/6 (16,667%) delle volte. Con un numero basso di lanci (per esempio 60 o 600), vedrete che generalmente le frequenze non sono distribuite in maniera uniforme. Aumentando il numero dei lanci di dado (per esempio 60.000), vedrete che le frequenze diventano molto più simili tra loro. Quando poi simulerete un numero significativo di lanci (per esempio 60.000.000), le barre sembreranno della stessa lunghezza.

I giochi e la simulazione descritti usano differenti funzionalità di raylib, tra cui forme, colori, suoni, animazione, rilevamento delle collisioni ed eventi con input da parte dell’utente (come fare clic con il mouse).

Dopo aver studiato il nostro codice, userete le funzionalità di raylib apprese (grafica, animazione e suono) per migliorare la vostra implementazione della gara tra la tartaruga e la lepre del Capitolo 5. Vi aggiungerete i suoni di una tipica corsa di cavalli nonché numerose immagini della lepre e della tartaruga per creare una divertente “rappresentazione spettacolare” multimediale animata. Userete poi raylib per migliorare la simulazione ad alte prestazioni di questo capitolo per mescolare e distribuire carte da gioco in modo che visualizzi le immagini delle carte. Infine, potrete scegliere tra 10 esercizi supplementari di programmazione e simulazione di giochi con raylib. Siate creativi: divertitevi anche a progettare e creare i vostri giochi!

Capitolo 11 - Caso pratico: costruire un sistema per l’elaborazione di transazioni usando file ad accesso casuale

In questo caso pratico userete l’elaborazione di file ad accesso casuale per implementare un semplice sistema per l’elaborazione di transazioni che simuli il tipo di funzionalità ad accesso diretto, ad alta velocità, che si ritrovano nei sistemi di gestione di database a livello industriale. Questo caso pratico vi permette di sperimentare sia la programmazione di applicazioni sia la programmazione di sistemi “*under the hood*”, ovvero “dietro le quinte”.

Capitolo 11 - Caso pratico sull’intelligenza artificiale: elaborazione del linguaggio naturale (NLP)

L’elaborazione del linguaggio naturale (NLP, *Natural Language Processing*) consente ai computer di comprendere, analizzare ed elaborare testi. Uno dei suoi utilizzi più comuni è l’“analisi del sentimento”, mediante la quale si determina se il testo ha una caratterizzazione positiva, neutrale o negativa. Un altro uso interessante dell’NLP è la valutazione del grado di leggibilità del testo, che è influenzato da tipo di vocaboli, lunghezza delle parole, struttura e lunghezza delle frasi, argomento e altri fattori. Nello scrivere questo libro abbiamo usato la versione a pagamento dello strumento Grammarly²² (NPL) per mettere a punto la stesura del testo e assicurarci che fosse leggibile per una vasta platea di lettori. I docenti che usano il metodo “flipped classroom” preferiscono usare manuali che gli studenti possano comprendere per conto proprio.

Alcuni studiosi ritengono che le opere di William Shakespeare siano in realtà state scritte da Christopher Marlowe, da Sir Francis Bacon o da altri ancora.^{23,24} Nell’esercizio sull’NLP userete le tecniche di elaborazione di array, stringhe e file per eseguire un semplice rilevamento delle similarità tra le opere in lingua originale *Romeo and Juliet* di Shakespeare ed *Edward the Second* di Marlowe.

Capitolo 11 - Caso pratico sull’intelligenza artificiale: machine learning con la Libreria Scientifica GNU

Le statistiche possono essere ingannevoli. Dataset estremamente diversi potrebbero avere statistiche descrittive identiche, o quasi. Esaminerete un famoso esempio di questo fenomeno, il quartetto di Anscombe,²⁵ che com-

22. Grammarly ha una versione gratuita e una a pagamento (<https://www.grammarly.com>). Vengono forniti plug-in gratuiti che si possono utilizzare in diversi popolari browser web.

23. “Did Shakespeare Really Write His Own Plays?” Accesso 13 novembre 2020. <https://www.history.com/news/did-shakespeare-really-write-his-own-plays>.

24. “Shakespeare authorship question.” Accesso 13 novembre 2020. https://en.wikipedia.org/wiki/Shakespeare_authorship_question.

25. “Anscombe’s quartet.” Accesso 13 novembre 2020. https://en.wikipedia.org/wiki/Anscombe%27s_quartet.

prende quattro dataset con coppie di coordinate *x-y* molto diverse, ma che hanno statistiche descrittive praticamente identiche. Studierete quindi un **esempio con codice completo** che usa la tecnica di **machine learning** chiamata **regressione lineare semplice** per calcolare l'equazione di una linea retta ($y = mx + b$) che, data una collezione di punti (coppie di coordinate *x-y*) che rappresentano una *variabile indipendente* (*x*) e una *variabile dipendente* (*y*), descrive la relazione tra queste variabili con una linea retta, nota come retta di regressione. Come potrete constatare, le rette di regressione sono visivamente identiche per tutti e quattro i diversi dataset del quartetto di Anscombe. Il programma che studierete passa quindi i comandi al **pacchetto open source gnuplot** in modo da creare diverse interessanti **visualizzazioni**. Poiché gnuplot usa per disegnare un proprio linguaggio differente dal C, nel nostro codice includiamo commenti esaustivi che ne spiegano i comandi. Infine, il caso pratico vi chiede di **eseguire una regressione lineare semplice su 126 anni di dati relativi alle temperature medie di gennaio a New York City, per determinare se la tendenza è verso il riscaldamento oppure il raffreddamento**. Inoltre, sempre in questo caso pratico, leggerete file di testo con valori separati da virgole (CSV) contenenti i dataset.

Capitolo 11 - Servizi web e il cloud: ottenere un bollettino meteorologico utilizzando libcurl e i servizi web OpenWeatherMap; introduzione ai mashup

Sempre più operazioni informatiche vengono ormai eseguite “nel cloud”, utilizzando software e dati distribuiti in tutto il mondo attraverso Internet. Le applicazioni che usiamo ogni giorno sono largamente dipendenti da diversi servizi basati sul cloud. Un servizio a cui si può accedere tramite Internet è detto **servizio web**. In questo caso pratico presentato negli esercizi lavorerete con un'applicazione con codice completo, che usa la **libreria open source libcurl in C** per invocare un servizio web **OpenWeatherMap** (livello gratuito) che restituisce la situazione meteo attuale per una determinata città. Il servizio web restituisce i risultati in formato **JSON (JavaScript Object Notation)**, che elaboriamo usando la **libreria open source cJSON**.

Con questo esercizio si spalanca un mondo di possibilità. Potrete esplorare circa 24.000 servizi web, elencati nella directory di **ProgrammableWeb**.²⁶ Molti sono gratuiti, o offrono livelli gratuiti che potrete usare per creare **mashup** divertenti e interessanti attraverso la combinazione di servizi web complementari.

Capitolo 12 - Caso pratico sul software dei sistemi: costruire il proprio compilatore

Nel corso di una serie di esercizi, **costruirete un semplice compilatore** che converte in linguaggio macchina del Simpletron (SML) programmi scritti in un semplice linguaggio di programmazione di alto livello. **Scriverete programmi in questo nuovo piccolo linguaggio di alto livello, li compilerete sul compilatore da voi costruito e li eseguirete sul vostro simulatore Simpletron**. E come spiegato nel **Capitolo 11**, il vostro compilatore può scrivere il codice generato in linguaggio macchina in un file dal quale il vostro computer Simpletron potrà leggere il vostro programma in SML, **caricarlo** in memoria ed eseguirlo! Questa serie completa di esercizi è molto utile per gli studenti di informatica meno esperti.

Capitolo 13 - Visualizzazione dell'algoritmo di ordinamento per fusione ad alte prestazioni

Nell'ambito del discorso sull'ordinamento, la nostra implementazione dell'**algoritmo di ordinamento per fusione (merge sort)** ad **alte prestazioni** rappresenta un elemento centrale. In questo caso pratico userete gli output per **visualizzare** i passi di partizionamento e di fusione dell'algoritmo, che vi aiuteranno a capire come funziona l'ordinamento per fusione.

Appendice C - Caso pratico sull'architettura dei sistemi: prestazioni con sistemi multithreading e multicore

Il **multithreading** è una tecnica che consente di suddividere un programma in “**thread**” separati, che possono essere eseguiti in **parallelo**. Risale ad alcuni decenni fa, ma ultimamente ha suscitato un grande interesse dovuto alla disponibilità di **processori multicore** nei computer e in altri dispositivi, tra cui smartphone e tablet. Questi processori implementano in modo economico più processori su un unico circuito integrato. Fanno eseguire in parallelo a ciascun core una parte differente del vostro programma, rendendo così più veloce il processo di completamento delle singole attività e del programma nel suo insieme. In molti dispositivi odierni vi sono comunemente quattro o otto core, e il loro numero continua ad aumentare. Per scrivere e testare il codice

26. “ProgrammableWeb.” Accesso 22 novembre 2020. <https://programmableweb.com/>.

di questo libro abbiamo usato un MacBook Pro a otto core. Le **applicazioni multithread** consentono di eseguire contemporaneamente thread separati su più core, in modo da sfruttare al meglio i vantaggi dell'architettura multicore.

Presentiamo un caso pratico con due diversi programmi che dimostrerà in maniera convincente la potenza del multithreading su un sistema multicore. Un programma esegue due calcoli molto complessi **in sequenza**, l'altro esegue i medesimi calcoli **in thread paralleli**. Abbiamo cronometrato il tempo totale di esecuzione necessario a svolgere questi calcoli per ciascun programma. I risultati mostrano che i tempi diminuiscono notevolmente quando la versione multithread del programma viene eseguita su un sistema multicore.

Programmazione sicura in C

Nelle linee guida di ACM/IEEE per i programmi di studio viene sottolineata l'importanza della sicurezza: il termine **è citato 395 volte nella documentazione per i corsi in informatica e 235 volte in quella per i corsi di tecnologie dell'informazione**. Nel 2017 sono state pubblicate le indicazioni per i **programmi di cybersecurity**, focalizzati sull'importanza della sicurezza sia nei corsi dedicati all'argomento specifico che negli altri corsi di informatica. In questa documentazione **il termine “sicurezza” è citato 865 volte**.

Alla fine dei Capitoli 2-12 e del Capitolo 14 troverete un paragrafo dedicato alla **Programmazione sicura in C**, il cui scopo è coinvolgere gli studenti che si avvicinano alla programmazione nelle problematiche che possono provocare violazioni della sicurezza. Questi paragrafi presentano alcune questioni e tecniche chiave e forniscono link e rimandi, così che possiate proseguire nell'apprendimento. Il nostro obiettivo è di stimolarvi a prendere da subito in considerazione il tema della sicurezza, anche se questo è il vostro primo corso di programmazione.

L'esperienza ha dimostrato come sia molto difficile costruire sistemi informatici a livello industriale in grado di resistere agli attacchi. Oggi, via Internet, attacchi del genere possono essere istantanei e di portata globale. **Le vulnerabilità del software derivano spesso da semplici problemi di programmazione**. Includere aspetti relativi alla sicurezza nel software dall'inizio del ciclo di sviluppo può ridurre sensibilmente le vulnerabilità.

La Divisione CERT del Software Engineering Institute della Carnegie Mellon University

<https://www.sei.cmu.edu/about/divisions/cert/index.cfm>

è stata creata per analizzare e rispondere prontamente agli attacchi informatici. Pubblica e promuove standard di codifica sicura per aiutare i programmati in C (e non solo) a implementare sistemi a livello industriale evitando pratiche di programmazione che espongono i sistemi agli attacchi. Gli standard CERT si evolvono quando sorgono nuovi problemi di sicurezza.

Vi spieghiamo come aggiornare il vostro codice (in modo adatto per un libro introduttivo) per conformarlo alle più recenti raccomandazioni di codifica sicura in C. Se state realizzando sistemi in C in ambito industriale, vi suggeriamo di leggere le norme contenute nel **SEI CERT C Coding Standard**, consultabili sul sito

<https://wiki.sei.cmu.edu/confluence/display/c>

Robert Seacord, un revisore tecnico di un'edizione precedente di questo libro, ha fornito raccomandazioni specifiche per ognuno dei nostri paragrafi sulla programmazione sicura in C. All'epoca ricopriva il ruolo di Secure Coding Manager al CERT e professore aggiunto alla School of Computer Science della Carnegie Mellon University. Attualmente è direttore tecnico della NCC Group, una società che si occupa di sicurezza informatica.

I nostri paragrafi sulla programmazione sicura in C prendono in esame molti argomenti importanti, tra cui:

- il controllo degli overflow aritmetici;
- le funzioni più sicure nell'**Annex K del C standard**;
- l'importanza del controllo delle informazioni sullo stato restituite dalle funzioni della Libreria Standard;
- il controllo degli intervalli;
- la generazione sicura di numeri casuali;
- il controllo dei confini degli array;
- la prevenzione degli overflow dei buffer;

- la validazione dell'input;
- le tecniche per evitare comportamenti indefiniti;
- la scelta di funzioni che restituiscono informazioni sullo stato anziché di funzioni simili che non lo fanno;
- la garanzia che i puntatori siano sempre NULL o contengano indirizzi validi;
- l'uso delle funzioni del C rispetto all'uso delle macro del preprocessore, e altro ancora.

Ottenere il codice degli esempi e installare il software

Per vostra comodità, forniamo tutti gli esempi del libro come file di codice sorgente in linguaggio C (.c), da utilizzare con ambienti di sviluppo integrati (IDE, *Integrated Development Environments*) e compilatori della riga di comando. Consultate la sezione “Prima di cominciare”, che segue la Prefazione, per istruzioni dettagliate sull’installazione del software. Nei test guidati del Capitolo 1 troverete le informazioni necessarie per eseguire il codice degli esempi del libro. Per qualsiasi problema, potete contattarci all’indirizzo e-mail deitel@deitel.com oppure compilando l’apposito modulo sul nostro sito <https://deitel.com/contact-us>.

Docker

Introduciamo **Docker**, uno strumento per “impacchettare” il software in **contenitori (container)** nei quali è compreso tutto ciò che è necessario per eseguirlo in maniera conveniente, **riproducibile e portabile** su varie piattaforme. L’installazione e la configurazione di alcuni pacchetti software sono piuttosto complicate. Nella maggior parte di questi casi potrete scaricare **contenitori Docker** preesistenti che vi consentiranno di evitare le complesse problematiche di installazione. Potrete semplicemente eseguire il software localmente sul vostro computer desktop o portatile, usando Docker come un ottimo strumento per iniziare a utilizzare nuove tecnologie in modo rapido, conveniente ed economico. Per vostra comodità, vi mostriamo come installare ed eseguire un contenitore Docker **preconfigurato con la GNU Compiler Collection (GCC)**, che include il compilatore **gcc**. Può essere eseguito in Docker su **Windows**, **macOS** e **Linux**, ed è particolarmente utile per coloro che utilizzano Visual C++, linguaggio in grado di compilare codice C ma non conforme al 100% con il C standard più aggiornato.

Docker è utile anche per quanto riguarda la **riproducibilità**. Potete configurare i contenitori Docker personalizzati con qualsiasi pezzo di software e qualsiasi libreria che utilizzate. Questo consentirà ad altri di ricreare l’ambiente che avete usato e riprodurre quindi il vostro lavoro, permettendo anche a voi di riprodurre i vostri risultati. La **riproducibilità** è particolarmente importante nel campo scientifico e medico, per esempio quando i ricercatori vogliono dimostrare e condividere il loro lavoro tramite la pubblicazione di articoli.

Flipped classroom

Oggigiorno, molti docenti usano l’approccio detto “**flipped classroom**” o “classe capovolta”.^{27,28} Gli studenti imparano l’argomento per conto proprio prima della lezione, e il tempo in aula viene usato per esercizi di programmazione, lavori di gruppo e discussioni. Il nostro libro con i suoi supplementi è adatto per questo metodo di apprendimento.

- Abbiamo usato **Grammarly** per controllare il livello di lettura del libro e verificare che fosse appropriato per l’apprendimento autonomo.
- Parallelamente alla lettura del testo, gli studenti dovrebbero eseguire i **147 esempi di codice in C** “dal vivo” e gli **oltre 350 esercizi di autovalutazione con risposte** per un riscontro immediato. In questo modo viene incoraggiata la partecipazione attiva degli studenti, che imparano i contenuti un passo per volta, con il metodo “**leggere, fare, verificare**”, adatto per il tipo di apprendimento attivo, autodidattico e pratico delle flipped classroom.

27. https://en.wikipedia.org/wiki/Flipped_classroom.

28. <https://www.edsurge.com/news/2018-05-24-a-case-for-flipping-learning-without-videos>.

- Proponiamo **445 esercizi e progetti**, che possono essere svolti dagli studenti a casa e/o in classe. Molti esercizi sono di livello elementare o intermedio, e possono quindi essere eseguiti in autonomia; inoltre in molti casi sono adatti per **progetti di gruppo** da sviluppare in classe.
- Includiamo in quasi tutti i capitoli un **riepilogo per paragrafo** con i termini chiave in **grassetto** per una veloce rassegna dei contenuti.
- Nell'ampio indice analitico, sono evidenziati in **grassetto** i numeri delle pagine contenenti le **definizioni** dei termini chiave, rendendo semplice per gli studenti trovare l'introduzione all'argomento che stanno studiando. In questo modo si facilita l'apprendimento autonomo caratteristico delle **flipped classroom**.

Un aspetto fondamentale delle flipped classroom è riuscire a ottenere le risposte alle domande che ci vengono poste quando lavoriamo per conto nostro.

Approccio didattico

Il libro contiene una ricca raccolta di esempi, esercizi, progetti e casi pratici tratti da molti ambiti diversi. Gli studenti dovranno risolvere interessanti **problemi del mondo reale** lavorando con **dati del mondo reale**. L'attenzione viene posta in particolare su una buona **ingegneria del software** e sull'importanza della **chiarezza dei programmi**.

Utilizzo dei font per conferire enfasi

I termini più importanti e i numeri di pagina nell'indice analitico in cui se ne trova la definizione sono evidenziati in **grassetto**. Per il codice C useremo il carattere Andale Mono (per esempio, `x = 5`), mentre per le scritte che appaiono sullo schermo useremo il carattere **Helvetica Neue LT Std in grassetto** (per esempio, il menu **File**).

Evidenziazione del testo in base alla sintassi

Per migliorare la leggibilità, il codice è evidenziato in base alla sintassi. Le nostre convenzioni di evidenziazione sono:

i commenti appaiono in questo modo
le parole chiave appaiono in questo modo
le costanti e i valori letterali appaiono in questo modo
 tutto il resto del codice appare in questo modo

Obiettivi dei capitoli

Ogni capitolo inizia con una lista di obiettivi per indicarvene il contenuto e darvi l'opportunità, alla fine della lettura, di determinare se li avete raggiunti tutti.

Esempi

Ci sono 147 esempi di codice “dal vivo” con migliaia di righe di codice verificato.

Tabelle e illustrazioni

Il libro include un gran numero di tabelle e disegni.

Esperienza nella programmazione

La trattazione degli argomenti è stata integrata con il sapere degli autori, che assieme possiedono nove decadi di esperienza sia per quanto riguarda la programmazione che l'insegnamento, e con le osservazioni dei professionisti in campo industriale e accademico che hanno recensito le nove edizioni di questo libro negli ultimi 29 anni. Troverete quanto segue:

- **Buone pratiche di programmazione** ed espressioni preferibili in linguaggio C che vi aiuteranno a creare programmi più chiari, più comprensibili e più semplici da mantenere.
- **Errori comuni di programmazione** per ridurre la possibilità di compiere tali errori.



- **Suggerimenti per prevenire errori** con consigli per trovare i bug e rimuoverli dai vostri programmi. Molti di questi suggerimenti indicano le tecniche per prevenire, in primo luogo, la comparsa di bug nei vostri programmi.
- **Suggerimenti per le prestazioni** che evidenziano le possibilità di aumentare la velocità di esecuzione dei vostri programmi o ridurre al minimo lo spazio di memoria occupato.
- **Osservazioni di ingegneria del software** che danno risalto agli aspetti architetturali e progettuali inerenti alla costruzione di sistemi software, specialmente per sistemi a larga scala.
- **Pratiche ottimali per la sicurezza** che vi aiuteranno a rinforzare i vostri programmi contro possibili attacchi informatici.

Riepilogo di ogni capitolo

Alla fine di ogni capitolo c'è la sezione "Riepilogo" che può contenere una sintesi dei contenuti dei singoli paragrafi, esercizi e casi pratici.

Software gratuito utilizzato nel libro

La sezione "Prima di cominciare", che segue questa Prefazione, spiega come installare il software che vi servirà per lavorare con i nostri esempi. Abbiamo testato gli esempi del libro utilizzando i seguenti noti compilatori scaricabili gratuitamente:

- **GNU gcc** su Linux, che è già installato su gran parte dei sistemi Linux e può essere installato sui sistemi macOS e Windows.
- **Visual Studio Community Edition** di Microsoft su Windows.
- **Compilatore Clang** di Apple in Xcode su macOS.

GNU gcc in Docker

Mostriamo inoltre il compilatore **GNU gcc** in un **contenitore Docker**, ideale per quei docenti che vogliono far utilizzare GNU gcc a tutti i loro studenti, a prescindere dai sistemi operativi in uso. In questo modo si offre agli utenti di Visual C++ un'opzione concreta di compilatore C, dal momento che Visual C++ non è 100% compatibile con il C standard più recente.

Il sottosistema Windows per Linux

Il **sottosistema Windows per Linux (WSL, Windows Subsystem for Linux)** permette agli utenti Windows di installare Linux ed eseguirlo in Windows. Vi indicheremo un link con le istruzioni passo per passo fornite da Microsoft per la configurazione di WSL e l'installazione di una distribuzione Linux, offrendo così un'ulteriore opzione agli utenti Windows per accedere al compilatore GNU gcc.

Documentazione sul C

Durante la lettura del libro, vi sarà utile la seguente documentazione.

- Manuale di riferimento della Libreria Standard del C del progetto GNU:
<https://www.gnu.org/software/libc/manual/pdf/libc.pdf>
- Manuale di riferimento sul linguaggio C:
<https://en.cppreference.com/w/c>
- File di intestazione della Libreria Standard del C:
<https://en.cppreference.com/w/c/header>
- Manuale di riferimento Microsoft sul linguaggio C:
<https://docs.microsoft.com/en-us/cpp/c-language/c-language-reference>

Comunicare con gli autori

Per domande, richieste di assistenza o per segnalare un errore potrete facilmente contattarci all'indirizzo

deitel@deitel.com

oppure tramite il modulo di contatto dal sito

<https://deitel.com/contact-us>

Potete interagire con noi attraverso i social media su

- **Facebook®** (<https://facebook.com/DeitelFan>)
- **Twitter®** (@deitel oppure <https://twitter.com/deitel>)
- **LinkedIn®** (<https://linkedin.com/company/deitel-&-associates>)
- **YouTube®** (<https://youtube.com/DeitelTV>)

I prodotti Deitel/Pearson su O'Reilly Online Learning

Gli iscritti a O'Reilly Online Learning possono accedere a molti libri di testo e professionali Deitel/Pearson, ai video LiveLessons e ai webinar giornalieri Full Throttle. Iscrivetevi per un periodo di prova gratuito di 10 giorni a

<https://deitel.com/LearnWithDeitel>

Libri di testo e professionali

Ogni e-book Deitel su O'Reilly Online Learning è in formato a colori e dettagliatamente indicizzato.

Corsi video LiveLessons asincroni

Imparerete in modo pratico con Paul Deitel, che vi presenta le avvincenti tecnologie informatiche all'avanguardia di Python, Python Data Science/AI e Java, oltre a video relativi a C++20 e C.

Webinar dal vivo Full Throttle

Paul Deitel offre **webinar Full Throttle** su O'Reilly Online Learning. Si tratta di corsi che durano una giornata intera, a ritmo serrato e ricchissimi di codice, con funzione introduttiva a Python, Python Data Science/AI, Java, C++20 e C. I webinar Full Throttle di Paul sono indicati per sviluppatori con esperienza e per i project manager che devono prepararsi per progetti che usano altri linguaggi di programmazione. Spesso i partecipanti ai corsi Full Throttle proseguono con i corsi video LiveLessons corrispondenti, che offrono l'opportunità di approfondire l'argomento con molte ore di apprendimento a ritmo meno sostenuto.

Ringraziamenti

Vorremmo ringraziare Barbara Deitel per le lunghe ore di ricerca su Internet dedicate a questo progetto. Siamo fortunati ad aver lavorato con la squadra di professionisti dell'editoria della Pearson. Abbiamo apprezzato la consulenza, l'esperienza e l'energia di Tracy Johnson (Pearson Education, Global Content Manager, Computer Science) in tutte le nostre pubblicazioni accademiche, sia in formato cartaceo che digitale. Ci ha spronato in ogni fase del processo a trovare le soluzioni giuste per creare i migliori libri. Carol Snyder ha gestito la produzione del libro e interagito con il team della Pearson che si occupa delle licenze, ottenendo rapidamente le autorizzazioni per grafiche e citazioni e permettendoci così di rispettare le tempistiche. Erin Sullivan ha reclutato e gestito il team dei revisori del libro. Noi abbiamo scelto l'immagine, e Chuti Prasertsith ha progettato la copertina, con il suo tocco magico nella realizzazione grafica.

Desideriamo ringraziare i nostri revisori, sia accademici che professionisti, per il loro impegno. Rispettando una tempistica serrata, i revisori hanno analizzato il testo, fornendo innumerevoli suggerimenti per migliorare la precisione, la completezza e la puntualità della presentazione. Hanno contribuito a creare un libro migliore.

Revisori**Revisori dell'ottava edizione**

Dr. Danny Kalev (Ben-Gurion University of the Negev, analista di sistema certificato, esperto del C ed ex membro del C++ Standards Committee) José Antonio González Seco (Parlamento dell'Andalusia)

Revisori della nona edizione

Dr. Brandon Invergo (GNU/European Bioinformatics Institute)

Jim Hogg (Program Manager, C/C++ Compiler Team, Microsoft Corporation)

José Antonio González Seco (Parlamento dell'Andalusia)

Alan Bunning (Purdue University)

Paul Clingan (Ohio State University)

Michael Geiger (University of Massachusetts, Lowell)

Dr. Danny Kalev (Ben-Gurion University of the Negev, analista di sistema certificato, esperto del C ed ex membro del C++ Standards Committee)

Jeonghwa Lee (Shippensburg University)

Susan Mengel (Texas Tech University)

Judith O'Rourke (SUNY di Albany)

Chen-Chi Shin (Radford University)

Revisori di altre recenti edizioni (affiliazione al tempo della revisione)

William Albrecht (University of South Florida)

Ian Barland (Radford University)

Ed James Beckham (Altera)

John Benito (Blue Pilot Consulting, Inc. e Coordinatore di ISO WG14 — il gruppo di lavoro responsabile del C Programming Language Standard)

Dr. John F. Doyle (Indiana University Southeast)

Alireza Fazelpour (Palm Beach Community College)

Mahesh Hariharan (Microsoft)

Hemanth H.M. (Software Engineer presso SonicWALL)

Kevin Mark Jones (Hewlett Packard)

Lawrence Jones, (UGS Corp.)

Don Kostuch (consulente indipendente)

Vytautas Leonavicius (Microsoft)

Xiaolong Li (Indiana State University)

William Mike Miller (Edison Design Group, Inc.)

Tom Rethard (The University of Texas di Arlington)

Robert Seacord (Secure Coding Manager presso SEI/

CERT, autore di *The CERT C Secure Coding Standard* ed esperto tecnico del gruppo di lavoro internazionale sulla standardizzazione del linguaggio di programmazione C)

Benjamin Seyfarth (University of Southern Mississippi)

Gary Sibbitts (St. Louis Community College di Meramec)

William Smith (Tulsa Community College)

Douglas Walls (Senior Staff Engineer, C compiler, Sun Microsystems — ora Oracle).

Un ringraziamento particolare per aver risposto cortesemente alle nostre domande va ad Alison Clear, Professore Associato presso la School of Computing dell'Eastern Institute of Technology (EIT) in Nuova Zelanda e copresidente della *Computing Curricula 2020 (CC2020) Task Force*, che ha rilasciato di recente le nuove linee guida per i corsi di informatica:

Computing Curricula 2020: Paradigms for Future Computing Curricula
<https://cc2020.nsparc.msstate.edu/wp-content/uploads/2020/11/Computing-Curricula-Report.pdf>

Per finire, uno speciale ringraziamento all'enorme numero di persone in tutto il mondo che, con le loro competenze tecnologiche, contribuiscono al movimento open source e condividono online il loro lavoro, nonché alle loro organizzazioni che incoraggiano il diffondersi di informazioni e open software, e a Google, il cui motore di ricerca risponde al nostro flusso costante di domande, in una frazione di secondo, a ogni ora del giorno e della notte, e gratuitamente.

E con questo abbiamo concluso! Quando leggerete il libro, ci farebbe piacere ricevere i vostri commenti, critiche, correzioni e suggerimenti per migliorare. Potrete inviare tutte le vostre comunicazioni, domande incluse, al seguente indirizzo

deitel@deitel.com

Vi risponderemo prontamente. Benvenuti nell'emozionante mondo della programmazione in linguaggio C per gli anni 2020. Speriamo che abbiate un'esperienza di apprendimento informativa, divertente e stimolante con questa nona edizione, e che apprezziate questo sguardo sullo sviluppo software all'avanguardia con il C. Vi auguriamo grandi successi!

Paul Deitel

Harvey Deitel

Gli autori

Paul J. Deitel, CEO e Chief Technical Officer della Deitel & Associates, Inc., si è laureato presso il MIT e ha più di 41 anni di esperienza nel campo informatico. Avendo insegnato a sviluppatori software professionisti sin dal 1992, Paul è uno dei docenti di programmazione tra i più esperti al mondo. Ha tenuto centinaia di corsi di programmazione a clienti aziendali, accademici, governativi e militari, a livello internazionale, quali UCLA, Cisco, IBM, Siemens, Sun Microsystems (ora Oracle), Dell, Fidelity, NASA (Kennedy Space Center), National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, Nortel Network, Puma, iRobot e molti altri. Lui e il suo coautore, Dr. Harvey M. Deitel, sono tra i migliori autori al mondo di libri di testo sui linguaggi di programmazione, di libri professionali, video e produzioni interattive multimediali per e-learning, oltre a svolgere attività di formazione virtuali e in presenza.

Il **Dr. Harvey M. Deitel**, Chairman e Chief Strategy Officer della Deitel & Associates, Inc., ha 59 anni di esperienza nel campo informatico; ha conseguito la laurea di secondo livello in Ingegneria elettronica al MIT e il dottorato in Matematica presso la Boston University, seguendo in queste sedi studi di informatica (prima che venissero creati corsi di laurea specifici). Ha una vasta esperienza di insegnamento a livello universitario, e ha anche occupato la posizione di Direttore del Dipartimento di Informatica al Boston College prima di fondare nel 1991 la Deitel & Associates, Inc. con il figlio Paul. Le pubblicazioni Deitel hanno guadagnato un riconoscimento internazionale, con più di 100 traduzioni pubblicate in giapponese, tedesco, russo, spagnolo, francese, polacco, italiano, cinese (tradizionale e semplificato), coreano, portoghese, greco, urdu e turco. Il Dr. Deitel ha tenuto centinaia di seminari di formazione professionale presso clienti aziendali, accademici, governativi e militari.

Deitel® & Associates, Inc.

La Deitel & Associates, Inc., fondata da Paul Deitel e Harvey Deitel, è un'organizzazione riconosciuta a livello internazionale per l'authoring e la formazione aziendale, specializzata in linguaggi di programmazione, tecnologie a oggetti, sviluppo di applicazioni mobile e tecnologie software per Internet e il Web. La Deitel annovera tra i suoi clienti, per quanto riguarda la formazione, molte tra le più grandi aziende mondiali, agenzie governative, settori militari e istituzioni accademiche. L'azienda offre corsi di formazione con docenti, sia virtualmente che in presenza, presso le sedi dei clienti in tutto il mondo e per Pearson Education su O'Reilly Online Learning.

Grazie a 45 anni di partnership editoriale con Pearson/Prentice Hall, Deitel & Associates, Inc. pubblica libri di testo e professionali all'avanguardia sulla programmazione (sia in formato **cartaceo** che **e-book**), corsi video **LiveLessons**, **webinar** dal vivo su **O'Reilly Online Learning** e corsi universitari multimediali interattivi **Revel™**.

Per contattare la Deitel & Associates, Inc. e gli autori, o per richiedere una proposta formativa, potete usare l'indirizzo:

deitel@deitel.com

Per maggiori informazioni sui programmi di formazione aziendale, visitate:

<http://www.deitel.com/training>

Per registrarsi per un periodo di prova gratuito di 10 giorni con O'Reilly Online Learning, visitate:

<https://deitel.com/LearnWithDeitel>

che vi rimanderà alla nostra pagina di destinazione su O'Reilly Online Learning. In questa pagina, fate clic sul link **Begin a free trial**.

Deitel & Associates, Inc.
1000 Corporate Park Drive, Suite 100
Mahwah, NJ 07430 USA

Prima di cominciare

Come prima cosa, leggete attentamente questa sezione per capire le convenzioni da noi utilizzate e assicurarvi che il vostro computer sia in grado di compilare ed eseguire i nostri programmi di esempio.

Convenzioni sui caratteri e i nomi

Utilizziamo diversi formati di carattere per distinguere le componenti dell'applicazione e del codice in C dal semplice testo. Per le componenti che appaiono su schermo, come il menu **File**, useremo il carattere **Helvetica Neue LT Std in grassetto**. Per gli elementi del C utilizzeremo un carattere Andale Mono, come in `sqrt(9)`.

Ottenere il codice degli esempi

Il codice degli esempi (Code Examples) è disponibile nella piattaforma MyLab. Una volta completato il download del file .zip, assicuratevi di estrarre i contenuti.

Nelle nostre istruzioni, consideriamo che gli esempi si trovino nella sottocartella chiamata `examples` nella cartella `Documents` del vostro profilo.

Compilatori utilizzati

Abbiamo testato gli esempi del libro utilizzando i seguenti compilatori gratuiti:

- Per Microsoft Windows, abbiamo usato Visual Studio Community Edition di Microsoft¹, che include il compilatore Visual C++ e altri strumenti di sviluppo Microsoft. Visual C++ è in grado di compilare il codice in C.
- Per macOS, abbiamo usato Xcode di Apple, che include il compilatore Clang per C. La riga di comando Clang può essere installata anche su sistemi Linux e Windows.
- Per Linux, abbiamo usato il compilatore GNU `gcc`, che fa parte della GNU Compiler Collection (GCC). GNU `gcc` è già installato sulla maggior parte dei sistemi Linux e può essere installato su sistemi macOS e Windows.

In questa sezione vi spieghiamo come installare i compilatori. I test guidati del Paragrafo 1.10 vi mostrano come compilare ed eseguire i programmi utilizzando questi compilatori.

Installare Visual Studio Community Edition su Windows

Se utilizzate Windows, come prima cosa assicuratevi che il vostro sistema soddisfi i requisiti per supportare Visual Studio Community Edition di Microsoft, verificando su:

<https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements>

Andate poi all'indirizzo:

<https://visualstudio.microsoft.com/downloads/>

per eseguire l'installazione con la procedura qui descritta.

1. La versione corrente al momento della stesura del libro era la Visual Studio 2019 Community Edition.

1. Fate clic su **Free download** sotto **Community**.
2. A seconda del vostro browser, potrebbe apparire un pop-up nella parte inferiore del vostro schermo nel quale fare clic su **Run** per avviare il processo di installazione. Se non dovesse apparire, fate doppio clic sul file di installazione nella vostra cartella **Downloads**.
3. Nella finestra di dialogo **User Account Control**, fate clic su **Yes** per abilitare le modifiche al vostro sistema.
4. Nella finestra di dialogo **Visual Studio Installer**, fate clic su **Continue** per permettere il download degli elementi necessari alla configurazione dell'installazione.
5. Per gli esempi di questo libro, selezionate l'opzione **Desktop Development with C++**, che include il compilatore Visual C++ e le Librerie Standard del C e del C++.
6. Fate clic su **Install**. Il tempo di installazione potrebbe essere considerevole, ma dipende dalla velocità della vostra connessione Internet.

Installare Xcode su macOS

Per installare Xcode su macOS, osservate le seguenti istruzioni.

1. Fate clic sul menu Apple e selezionate **App Store...**, oppure fate clic sull'icona di **App Store** sul Dock nella parte inferiore dello schermo.
2. Nel campo **Cerca dell'App Store**, digitate **Xcode**.
3. Fate clic sul pulsante **Ottieni** per installare Xcode.

Installare GNU gcc su Linux

La maggior parte degli utenti Linux hanno già installata una versione recente di GNU gcc. Per verificare, apriete una shell o una finestra di Terminale sul vostro sistema Linux, quindi inserite il comando

```
gcc --version
```

Se il comando non viene riconosciuto, allora dovete installare GNU gcc. Noi usiamo la distribuzione Ubuntu Linux, sulla quale è necessario accedere come amministratore oppure avere la password di amministratore per poter eseguire i seguenti comandi:

1. `sudo apt update`
2. `sudo apt install build-essential gdb`

Spesso le diverse distribuzioni di Linux usano differenti software di installazione e tecniche di aggiornamento. Se non state utilizzando Ubuntu, cercate online “Installare GCC su *NomeDistribuzioneLinux*”, sostituendo *NomeDistribuzioneLinux* con il nome della vostra versione di Linux. Potrete scaricare la GNU Compiler Collection per varie piattaforme da:

```
https://gcc.gnu.org/install/binaries.html
```

Installare GNU GCC su Ubuntu Linux con il sottosistema Windows per Linux

Un altro modo per installare GNU gcc su Windows è tramite il **sottosistema Windows per Linux (WSL, Windows Subsystem for Linux)**, che vi consente di eseguire Linux su Windows. Ubuntu Linux fornisce in Windows Store un installer facilmente utilizzabile, ma dovete prima procedere all'installazione di WSL.

1. Nella casella di ricerca della barra delle applicazioni digitate “Attiva o disattiva funzionalità di Windows”, quindi fate clic su **Apri** nei risultati di ricerca.
2. Nella finestra di dialogo **Funzionalità di Windows**, individuate **Sottosistema Windows per Linux** e verificate se la relativa casella è spuntata. Se questo è il caso, allora WSL è già installato. In caso contrario, selezionate la casella e fate clic su **OK**. Windows provvederà a installare WSL e vi chiederà di riavviare il sistema.

3. Una volta riavviato il sistema ed effettuato l'accesso, aprite l'app **Microsoft Store** e cercate **Ubuntu**, selezionate l'applicazione **Ubuntu** e fate clic sul pulsante per avviare l'installazione. Verrà così installata la versione più recente di Ubuntu Linux.
4. A installazione avvenuta, fate clic sul pulsante di avvio del programma per visualizzare la finestra della riga di comando di Ubuntu Linux, che proseguirà il processo di installazione. Vi verrà chiesto di creare un nome utente e una password per Ubuntu, che non devono necessariamente corrispondere a quelli da voi utilizzati per Windows.
5. Quando sarà completata l'installazione di Ubuntu, eseguite i due comandi mostrati sotto per installare la GCC e il debugger GNU (potrete dover inserire la vostra password Ubuntu per l'account creato al punto 4):

```
sudo apt-get update
sudo apt-get install build-essential gdb
```

6. Verificate che gcc sia installato eseguendo il comando:

```
gcc --version
```

Per accedere ai nostri file di codice, usate il comando cd per modificare la cartella contenuta in Ubuntu in:

```
cd /mnt/c/Users/VostroNomeUtente/Documents/examples
```

Aggiornate con il vostro nome utente il percorso relativo alla posizione dei nostri esempi sul vostro sistema.

Contenitore Docker della GNU Compiler Collection (GCC)

Docker è uno strumento per “impacchettare” il software in **contenitori** (chiamati anche **immagini**) che raggruppano *tutti gli elementi* necessari per eseguire il software su diverse piattaforme. Docker è particolarmente utile nel caso di pacchetti software con installazioni e configurazioni complesse. Generalmente si possono scaricare gratuitamente contenitori Docker preesistenti (spesso si trovano su <https://hub.docker.com>) ed eseguirli localmente sul proprio computer desktop o portatile. Questo fa di Docker un ottimo mezzo per accostarsi alle nuove tecnologie in maniera rapida e conveniente.

Docker rende facilmente utilizzabile la GNU Compiler Collection sulla maggior parte delle versioni di Windows 10, oltre che su macOS e Linux. I contenitori Docker di GNU si trovano su

```
https://hub.docker.com/\_/gcc
```

Installare Docker

Per usare il contenitore Docker della GCC, è dapprima necessario installare Docker. Gli utenti Windows (a 64 bit)² e macOS dovranno scaricare ed eseguire l'installer **Docker Desktop** da:

```
https://www.docker.com/get-started
```

e seguire quindi le istruzioni sullo schermo. Gli utenti Linux dovranno installare **Docker Engine** da:

```
https://docs.docker.com/engine/install/
```

Registratevi inoltre per un account **Docker Hub** sulla seguente pagina web, in modo da poter installare contenitori preconfigurati scaricandoli da <https://hub.docker.com>.

2. Se avete Windows Home (a 64 bit), seguite le istruzioni all'indirizzo <https://docs.docker.com/docker-for-windows/install-windows-home/>.

Scaricare il contenitore Docker

Una volta che Docker è installato e operativo, apriete un prompt dei comandi (Windows), una finestra di Terminal (macOS/Linux) o una shell (Linux) ed eseguite il comando seguente:

```
docker pull gcc:latest
```

Docker eseguirà il download della versione corrente del contenitore della GNU Compiler Collection (GCC).³ In un test guidato del Paragrafo 1.10 vi mostreremo come avviare il contenitore e come usarlo per compilare ed eseguire programmi in linguaggio C.

3. Al momento della stesura del libro, la versione corrente della GNU Compiler Collection era la 10.2.

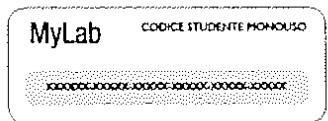
Pearson MyLab

UN AMBIENTE PER LO STUDIO

L'attività di apprendimento di questo corso continua in **MyLab**, l'**ambiente digitale per lo studio** che completa il libro offrendo **risorse didattiche** fruibili sia in **modo autonomo** sia per **assegnazione del docente**. Il **codice sulla copertina** di questo libro consente l'**accesso per 18 mesi a MyLab**.

COME ACCEDERE

1. **Registrati** come **studente universitario** all'indirizzo registrazione.pearson.it (Se sei già registrato passa al punto successivo);
2. effettua il **login** alla tua MyPearsonPlace all'indirizzo www.pearson.it/place e regista il prodotto digitale cliccando su **Attiva prodotto** ed inserendo il codice presente in copertina;
3. entra nella sezione *Prodotti* e clicca sul tasto **AVVIA** presente di fianco all'immagine della copertina del testo;
4. clicca su **classe MyLab studio autonomo** o, in alternativa, su **Iscriviti a una classe** ed inserisci il codice classe indicato dal tuo docente.



CHE COSA CONTIENE

MyLab offre la possibilità di accedere al Manuale online: l'**edizione digitale del testo** arricchita da funzionalità che permettono di personalizzarne la fruizione, attivare la sintesi vocale, inserire segnalibri.

Inoltre la **piattaforma digitale MyLab** integra e monitora il percorso individuale di studio con **attività formative e valutative specifiche**. La loro descrizione dettagliata è consultabile nella pagina di catalogo dedicata al libro, all'indirizzo link.pearson.it/F1966329 oppure tramite il presente QR code.



CAPITOLO

1

Sommario del capitolo

- 1.1 Introduzione
- 1.2 Hardware e software
- 1.3 Gerarchia dei dati
- 1.4 Linguaggi macchina, assembly e di alto livello
- 1.5 Sistemi operativi
- 1.6 Il linguaggio di programmazione C
- 1.7 Libreria Standard del C e librerie open source
- 1.8 Altri linguaggi di programmazione popolari
- 1.9 Un tipico ambiente di sviluppo per programmi in C
- 1.10 Test: eseguire un'applicazione in C negli ambienti Windows, Linus e macOS
- 1.11 Internet, World Wide Web, Cloud e IoT
- 1.12 Tecnologie software
- 1.13 Quanto sono grandi i big data?
- 1.14 Caso pratico: un'applicazione mobile di big data
- 1.15 Intelligenza artificiale, all'intersezione tra informatica e data science
- 1.16 Riepilogo

Introduzione ai computer e al linguaggio C

Obiettivi

- Conoscere i recenti ed emozionanti sviluppi nella programmazione.
- Apprendere i concetti base dell'hardware, del software e di Internet.
- Capire la gerarchia dei dati, dai bit alle basi di dati.
- Capire le diverse tipologie di linguaggi di programmazione.
- Capire i punti di forza del C e di altri linguaggi di programmazione.
- Introdurre alla Libreria Standard del C contenente funzioni riutilizzabili che aiutano a evitare di "reinventare la ruota".
- Testare un programma C con uno o più dei popolari compilatori C che abbiamo usato per sviluppare le centinaia di esempi di codice C, esercizi e progetti (EEP) del libro.
- Introdurre ai big data e alla data science.
- Introdurre all'intelligenza artificiale, intesa come intersezione chiave tra l'informatica e la data science.

1.1 Introduzione

Benvenuti nel linguaggio C, uno dei linguaggi di programmazione per computer più vecchi al mondo e, in base al Tiobe Index, al primo posto a livello di popolarità.¹ Probabilmente avrete già familiarità con le diverse attività svolte dai computer, ma questo manuale vi permetterà di avere una concreta e dettagliata esperienza nello scrivere istruzioni C che ordinano al computer di eseguirle.² Il **software** (ovvero le istruzioni C da voi scritte, chiamate anche **codice**) controlla l'**hardware** (ovvero il computer e i dispositivi a esso collegati).

Il linguaggio C è largamente usato a livello industriale per una vasta gamma di applicazioni. Sono infatti scritti in parte in C i sistemi operativi per desktop oggi più importanti (ovvero Windows³, macOS⁴ e Linux⁵), così come molte applicazioni famose, tra cui browser web (es. Google

1. "TIOBE Index." Accesso 4 novembre 2020. <https://www.tiobe.com/tiobe-index/>.

2. "After All These Years, the World is Still Powered by C Programming." Accesso 4 novembre 2020. <https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming>.

3. "What Programming Language is Windows written in?" Accesso 4 novembre 2020. <https://social-microsoft.com/Forums/en-US/65aife05-9c1d-48bf-bd40-148e6b3da9f1/what-programming-language-is-windows-written-in>.

4. "macOS." Accesso 4 novembre 2020. <https://en.wikipedia.org/wiki/MacOS>.

5. "Linux kernel." Accesso 4 novembre 2020. https://en.wikipedia.org/wiki/Linux_kernel.

Chrome⁶ e Mozilla Firefox⁷), sistemi di gestione di database (es. Microsoft SQL Server⁸, Oracle⁹ e MySQL¹⁰) ecc.

In questo capitolo intodurremo concetti e terminologia necessari alla programmazione in C che imparerete, a partire dal Capitolo 2. Intodurremo concetti basilari relativi a software e hardware e faremo una panoramica sulla gerarchia dei dati, dai singoli bit alle basi di dati che memorizzano l'enorme quantità di dati necessari all'implementazione di moderne applicazioni quali, per esempio, Google Search, Netflix, Twitter, Waze, Uber, Airbnb e molte altre.

Discuteremo le tipologie dei linguaggi di programmazione. Intodurremo la Libreria Standard del C e diverse librerie “open source” basate sul C che vi eviteranno di “reinventare la ruota”. Utilizzerete queste librerie per riuscire a svolgere attività significative con un numero modesto di istruzioni. Intodurremo inoltre tecnologie aggiuntive che vi aiuteranno nello sviluppo del software nella vostra carriera.

Sono disponibili numerosi ambienti di sviluppo che vi permettono di compilare, costruire ed eseguire applicazioni in C. Studierete uno o più dei quattro test guidati che vi mostrano come compilare ed eseguire codice in C utilizzando:

- Microsoft Visual Studio 2019 Community Edition per Windows;
- Clang in Xcode su macOS;
- GNU gcc in una shell su Linux;
- GNU gcc in una shell che viene eseguita all'interno del contenitore Docker della GNU Compiler Collection (GCC).

Potete leggere soltanto il test o i test necessari per il vostro corso o per il vostro progetto professionale.

Nel passato, la maggior parte dei programmi funzionava su singoli computer (non connessi tra loro). Le applicazioni moderne invece sono in grado di comunicare tra i computer di tutto il mondo attraverso Internet. Pertanto, intodurremo Internet, il World Wide Web, il cloud e l'Internet delle cose (IoT, *Internet of Things*), che potranno tutti svolgere un ruolo significativo nelle applicazioni che svilupperete nei prossimi anni.

1.2 Hardware e software

I computer possono prendere decisioni logiche ed eseguire calcoli assai più velocemente di un essere umano. Molti degli attuali personal computer e smartphone possono eseguire miliardi di calcoli in un secondo, più di quanto un essere umano possa fare nella sua intera vita. I **supercomputer** già ora svolgono *milioni di miliardi* di istruzioni al secondo! A dicembre 2020, Fugaku¹¹, della Fujitsu, era il supercomputer più veloce del mondo, in grado di eseguire 442 biliardi di calcoli al secondo (442 petaflop)¹². Per avere un'idea delle dimensioni in gioco, questo supercomputer può eseguire in un secondo quasi 58 milioni di calcoli per ogni persona sul pianeta!¹³ E le prestazioni dei supercomputer sono in continuo miglioramento.

I computer elaborano i dati eseguendo sequenze di istruzioni chiamate **programmi (software)**. Questi software guidano il computer a compiere azioni specificate dalle persone che scrivono i programmi, i **programmatori**.

Un computer è costituito da diversi dispositivi fisici, i quali prendono il nome di hardware (come per esempio la tastiera, lo schermo, il mouse, le memorie a stato solido, i dischi rigidi, i lettori DVD e i processori). Grazie al rapido sviluppo delle tecnologie software e hardware, i costi dei computer si stanno riducendo drasticamente. Computer che decenni fa occupavano intere stanze e costavano milioni di dollari, ora sono costituiti da un singolo chip di silicio più piccolo di un'unghia e dal costo di qualche euro. Paradossalmente, il silicio è

6. “Google Chrome.” Accesso 4 novembre 2020. https://en.wikipedia.org/wiki/Google_Chrome.

7. “Firefox.” Accesso 4 novembre 2020. <https://en.wikipedia.org/wiki/Firefox>.

8. “Microsoft SQL Server.” Accesso 4 novembre 2020. https://en.wikipedia.org/wiki/Microsoft_SQL_Server.

9. “Oracle Database.” Accesso 4 novembre 2020. https://en.wikipedia.org/wiki/Oracle_Database.

10. “MySQL.” Accesso 4 novembre 2020. <https://en.wikipedia.org/wiki/MySQL>.

11. “Top 500.” Accesso 24 dicembre 2020. https://en.wikipedia.org/wiki/TOP500#TOP_500.

12. “Flops.” Accesso 1 novembre 2020. <https://en.wikipedia.org/wiki/FLOPS>.

13. Per vedere fino a che punto sono arrivate le prestazioni di elaborazione, considerate questo: nei suoi primi giorni di elaborazione negli anni Sessanta, Harvey Deitel usava il Digital Equipment Corporation PDP-1 (<https://en.wikipedia.org/wiki/PDP-1>), che era in grado di eseguire solo 93.458 operazioni al secondo, e l'IBM 1401 (<http://www.ibm-1401.info/1401GuidePosterV9.html>), che eseguiva solo 86.957 operazioni al secondo.

uno dei materiali più comuni sulla Terra: è infatti uno degli elementi presenti nella sabbia. La tecnologia dei chip basati su silicio ha reso i computer così economici da renderli un bene di consumo.

1.2.1 Legge di Moore

Ragionevolmente, col passare degli anni ci si aspetta di spendere un po' di più per la maggior parte dei servizi e dei prodotti. Nel campo delle telecomunicazioni e dei computer è avvenuto l'opposto, specialmente per quanto riguarda l'hardware a supporto di queste tecnologie. Negli ultimi anni, il costo dell'hardware è sceso rapidamente.

Per decenni, ogni due anni circa, le capacità dei computer approssimativamente si sono raddoppiate senza costi aggiuntivi. Questo trend è chiamato **Legge di Moore**, dal nome di colui che l'ha identificato negli anni Sessanta del secolo scorso, Gordon Moore, co-fondatore di Intel, una delle aziende leader nella produzione di processori per computer e **sistemi embedded (incorporati)**, quali elettrodomestici smart, sistemi di sicurezza per la casa, robot, incroci stradali intelligenti ecc.

Secondo quanto riportato da alcune figure chiave delle aziende NVIDIA e Arm, che producono processori, la legge di Moore non è più valida.^{14,15} La capacità di elaborazione dei computer continua ad aumentare, ma si basa su nuovi tipi di processori, come quelli multicore (Paragrafo 1.2.2).

La legge di Moore e altre *osservazioni correlate* si applicano specialmente a quanto segue:

- quantità di memoria dei computer riservata ai programmi;
- quantità di memoria secondaria (per esempio dischi rigidi o a stato solido) usata per immagazzinare dati e programmi;
- velocità dei processori, ovvero la velocità alla quale vengono eseguiti i programmi.

Una crescita analoga si è avuta nel campo delle telecomunicazioni. I costi sono precipitati in conseguenza dell'intensa competizione, dovuta all'incremento di domanda per la **larghezza di banda** usata dalle comunicazioni (ovvero, la capacità di trasmettere informazioni). Non c'è un altro campo nel quale la tecnologia migliora così velocemente mentre i costi diminuiscono così rapidamente. Questo progresso fenomenale sta realmente favorendo una **rivoluzione informatica**.

1.2.2 Organizzazione dei computer

Al di là delle differenze nell'aspetto esteriore, possiamo immaginarc i computer come composti da diverse **unità logiche** o parti.

Unità di input

Questa è la parte "di acquisizione": ottiene informazioni (dati e programmi) dai **dispositivi di input** e le mette a disposizione di altre unità per la loro elaborazione. La maggior parte dell'input prodotto dall'utente è immessa attraverso tastiere, mouse o dispositivi touchscreen e touchpad. Altre modalità di input includono:

- comandi vocali;
- scansione di immagini e codici a barre;
- lettura da dispositivi di memorizzazione secondaria (come dischi a stato solido, dischi rigidi, dischi Blu-ray™ e memorie USB, chiamate anche "chiavette");
- ricezione di video da webcam;
- ricezione di informazioni sul proprio computer attraverso Internet (come quando si vedono video in streaming da YouTube® o si scaricano e-book da Amazon);
- ricezione di coordinate da dispositivi GPS;

14. "Moore's Law turns 55: Is it still relevant?" Accesso 2 novembre 2020. <https://www.techrepublic.com/article/moores-law-turns-55-is-it-still-relevant/>.

15. "Moore's Law is dead: Three predictions about the computers of tomorrow." Accesso 2 novembre 2020, <https://www.techrepublic.com/article/moores-law-is-dead-three-predictions-about-the-computers-of-tomorrow/>.

- informazioni di movimento e posizione fornite da **accelerometri** (dispositivi che forniscono l'accelerazione nelle direzioni su/giù, sinistra/destra e avanti/indietro) presenti negli smartphone o nei controller wireless per videogiochi (come quelli per Microsoft® Xbox®, Nintendo Switch™ e Sony® PlayStation®);
- comandi vocali provenienti da assistenti intelligenti come Apple Siri®, Amazon Alexa® e Google Home®.

Unità di output

Questa unità di “trasmissione” prende le informazioni elaborate dal computer e le invia a vari **dispositivi di output** per poterle utilizzare all'esterno del computer. La maggior parte delle informazioni in uscita dai computer viene oggi:

- visualizzata su schermi;
- stampata su carta (pratica deprecabile in ottica “green”);
- riprodotta come audio o video sugli smartphone, sui tablet, sui PC o sugli schermi giganti negli stadi;
- trasmessa attraverso Internet;
- usata per controllare altri dispositivi come le automobili senza conducente (e i **veicoli autonomi** in generale), i robot e gli elettrodomestici “intelligenti”.

Le informazioni sono poi tipicamente salvate su dispositivi di memorizzazione, come dischi a stato solido (SSD), dischi rigidi, dischi DVD o chiavette USB. Modalità più recenti di output sono le vibrazioni degli smartphone o dei controller per videogiochi, i dispositivi per la realtà virtuale come Oculus Rift®, Oculus Quest®, Sony® PlayStation® VR e Samsung Gear VR®, e i dispositivi di realtà mista come Magic Leap® One e Microsoft HoloLens™.

Unità di memoria

Questa unità di “immagazzinamento”, ad accesso rapido e con una capacità limitata, conserva le informazioni arrivate dall'unità di input, rendendole immediatamente disponibili per l'elaborazione quando richieste. Questa parte conserva anche le informazioni già elaborate fino a quando possono essere inviate a dispositivi di uscita tramite l'unità di output. Le informazioni conservate in questa unità sono **volatili**, ovvero si perdono quando il computer viene spento. L'unità di memoria viene solitamente chiamata **memoria, memoria principale o RAM (Random Access Memory)**. La memoria principale su computer di tipo desktop o notebook può raggiungere 128 GB di RAM, sebbene comunemente vada da 8 a 16 GB. GB è l'acronimo di gigabyte; un **gigabyte** corrisponde circa a un miliardo di byte. Un **byte** è composto da 8 bit. Un **bit** (abbreviazione di “binary digit”) prende valore 0 o 1.

Unità aritmetica e logica (ALU, Arithmetic-Logic Unit)

Questa unità di “produzione” esegue **calcoli** (come somme, sottrazioni, moltiplicazioni e divisioni) e prende decisioni (per esempio, confrontando due oggetti in memoria per determinare se sono uguali o no). Nei sistemi moderni, la ALU è un componente della prossima unità logica, la CPU.

Unità centrale di elaborazione (CPU, Central Processing Unit)

Questa unità “amministrativa” coordina e supervisiona le operazioni delle altre unità. La CPU dice

- all'unità di input quando le informazioni devono essere lette dall'unità di memoria;
- alla ALU quando le informazioni lette dalla memoria devono essere usate nei calcoli;
- all'unità di output quando mandare le informazioni dalla memoria ai vari dispositivi di output.

Quasi tutti i computer hanno **processori multicore**, i quali implementano più processori in un unico circuito integrato. Tali processori possono eseguire diverse operazioni simultaneamente. Un **processore dual-core** ha due CPU, un **processore quad-core** ne ha quattro e un **processore octa-core** ne ha otto. Alcuni processori Intel arrivano fino a 72 core.

Unità di memorizzazione secondaria

Questa è l'unità di "stoccaggio" a lungo termine e ad alta capacità. I programmi o i dati non utilizzati attivamente da altre unità sono normalmente posti su dispositivi di memoria secondaria finché non servono di nuovo, forse ore, giorni, mesi o persino anni dopo. Le informazioni su dispositivi di memoria secondaria sono **persistenti**: si conservano anche quando l'alimentazione del computer viene disattivata. Per accedere alle informazioni in memoria secondaria ci vuole molto più tempo che per quelle in memoria primaria, ma il costo per byte è molto inferiore. Esempi di dispositivi di memoria secondaria sono i dischi a stato solido (SSD), le chiavette USB, i dischi rigidi e i dispositivi di lettura/scrittura Blu-ray. I dischi di oggi possono contenere terabyte (TB) di dati. Un terabyte corrisponde a circa mille miliardi di byte. In genere i dischi rigidi di computer desktop e portatili memorizzano fino a 4 TB, e alcuni dei più recenti dischi rigidi di computer desktop arrivano fino a 20 TB.¹⁶ Il più grande SSD commerciale può contenere fino a 100 TB (e costa \$40.000).¹⁷

✓ Autovalutazione

1. (*Completare*) Per molti decenni, ogni uno o due anni, le capacità dei computer approssimativamente si sono raddoppiate senza costi aggiuntivi. Questo notevole andamento è spesso chiamato _____.

Risposta: Legge di Moore.

2. (*Vero/Falso*) Le informazioni salvate nell'unità di memoria sono persistenti, ovvero si conservano anche quando il computer viene spento.

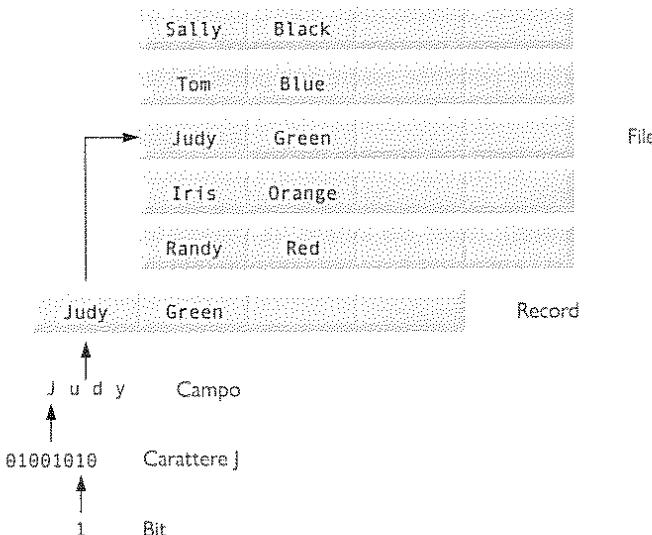
Risposta: Falso. Le informazioni conservative nell'unità di memoria sono volatili, ovvero vanno perse quando il computer viene spento.

3. (*Completare*) Molti computer hanno processori _____ che implementano più processori in un unico circuito integrato. Tali processori possono eseguire molte operazioni simultaneamente.

Risposta: multicore.

1.3 Gerarchia dei dati

Le informazioni elaborate dai computer formano una **gerarchia dei dati** che ha una struttura più grande e complessa man mano che si passa dai dati più semplici (chiamati "bit") a quelli più articolati, come i caratteri e i campi. Il seguente diagramma illustra una porzione della gerarchia dei dati:



16. "History of hard disk drives." Accesso 1 novembre 2020. https://en.wikipedia.org/wiki/History_of_hard_disk_drives.

17. "At 100TB, the world's biggest SSD gets an (eye-watering) price tag." Accesso 1 novembre 2020. <https://www.techradar.com/news/at-100tb-the-worlds-biggest-ssd-gets-an-eye-watering-price-tag>.

Bit

Un bit, abbreviazione di “*binary digit*”(una cifra che può assumere uno di *due* valori), è il più piccolo elemento di informazione in un computer e può avere valore 0 o 1. È sorprendente che le straordinarie attività svolte dai computer si riducano semplicemente a manipolare sequenze di 0 e 1: controllare il valore di un bit, impostare il valore di un bit e invertire il valore di un bit (da 1 a 0 o da 0 a 1). I bit formano la base del sistema numerico binario, che discuteremo nell’Appendice E (online), “Sistemi di numerazione”.

Caratteri

Lavorare con le informazioni nella loro forma elementare di bit è molto noioso. Le persone preferiscono piuttosto lavorare con **cifre decimali** (0-9), **lettere** (A-Z e a-z) e **simboli speciali** come

\$ @ % & * () - + " : ; , ? /

Cifre, lettere e simboli speciali sono chiamati **caratteri**. L’insieme di caratteri del computer è l’insieme di tutti i caratteri utilizzati per scrivere programmi e rappresentare dati. I computer elaborano soltanto le cifre binarie 1 e 0, quindi l’insieme di caratteri di un computer rappresenta ogni carattere come una combinazione di 1 e 0. Come impostazione predefinita, C utilizza l’insieme di caratteri **ASCII** (*American Standard Code for Information Interchange*). C supporta anche i caratteri **Unicode®** che sono composti ciascuno da uno, due, tre o quattro byte (8, 16, 24 o 32 bit, rispettivamente).¹⁸

Unicode contiene caratteri per molte diverse lingue. ASCII è un (minuscolo) sottoinsieme di Unicode che rappresenta lettere (a-z e A-Z), cifre e alcuni dei più comuni caratteri speciali. Potete consultare il sottoinsieme ASCII di Unicode all’indirizzo

<https://www.unicode.org/charts/PDF/U0000.pdf>

Potete visualizzare la tabella Unicode con tutte le lingue, i simboli, gli emoji e altro all’indirizzo

<http://www.unicode.org/charts/>

Campi

Così come i caratteri sono composti da bit, i **campi** sono composti da caratteri o da byte. Un campo è un gruppo di caratteri o di byte che esprime un significato. Per esempio, un campo composto da lettere maiuscole e minuscole può rappresentare il nome di una persona, un campo composto da cifre decimali può rappresentare l’età di una persona.

Record

Diversi campi correlati possono essere usati per creare un **record**. In un sistema per la gestione degli stipendi, per esempio, il record di un dipendente potrebbe essere composto dai seguenti campi (tra parentesi sono indicati i possibili tipi di questi campi):

- numero di matricola del dipendente (un numero intero);
- nome (un gruppo di caratteri);
- indirizzo (un gruppo di caratteri);
- paga oraria (un numero decimale);
- retribuzione annuale (un numero decimale);
- ammontare delle tasse trattenute (un numero decimale).

Quindi un record è un gruppo di campi in relazione tra loro. Nell’elenco precedente, tutti i campi si riferiscono al medesimo dipendente. Un’azienda può avere diversi dipendenti, ognuno con il suo record.

File

Un **file** è un gruppo di record correlati. Più in generale, un file contiene dati arbitrari codificati in maniera arbitraria. In alcuni sistemi operativi, un file è visto semplicemente come una *sequenza di byte*: qualsiasi struttura

18. “Programming with Unicode.” Accesso 1 novembre 2020. https://unicodebook.readthedocs.io/programming_languages.html.

dei byte nel file, come la suddivisione in record, è un'elaborazione creata dal programmatore. Nel Capitolo 11 vedrete come fare. Non è raro che in diversi frangenti la memorizzazione delle informazioni comporti l'utilizzo di numerosi file, alcuni dei quali contenenti miliardi o anche migliaia di miliardi di caratteri. Come vedremo di seguito, con i big data, è in continuo aumento il numero di file con dimensioni molto più grandi.

Database

Una **database** (**base di dati**) è una raccolta di dati organizzati in maniera tale da permetterne facilmente l'accesso e la manipolazione. Il modello più popolare è il **database relazionale**, nel quale i dati sono immagazzinati in semplici tabelle. Una tabella comprende record e campi. Per esempio, una tabella di studenti potrebbe comprendere nome, cognome, anno, numero di matricola dello studente e la media dei voti. I dati per ogni studente costituiscono un record e i singoli elementi di informazione in ogni record sono i campi. È possibile effettuare ricerche, ordinare e manipolare i dati in base alle loro relazioni in tabelle multiple o in più database. Per esempio, un'università potrebbe usare i dati del database degli studenti in combinazione con i database dei corsi, delle residenze universitarie, delle convenzioni per la mensa, ecc.

Big data

La tabella che segue mostra alcune comuni unità di misura dei byte:

Unità	Byte	Corrispondenza approssimativa
1 kilobyte (KB)	1.024 byte	10^3 (1.024) byte esattamente
1 megabyte (MB)	1.024 kilobyte	10^6 (1.000.000) byte
1 gigabyte (GB)	1.024 megabyte	10^9 (1.000.000.000) byte
1 terabyte (TB)	1.024 gigabyte	10^{12} (1.000.000.000.000) byte
1 petabyte (PB)	1.024 terabyte	10^{15} (1.000.000.000.000.000) byte
1 exabyte (EB)	1.024 petabyte	10^{18} (1.000.000.000.000.000.000) byte
1 zettabyte (ZB)	1.024 exabyte	10^{21} (1.000.000.000.000.000.000.000) byte

La quantità di dati prodotti nel mondo è enorme e in rapido aumento. Le applicazioni relative ai **big data** trattano imponenti quantità di dati. Questo campo cresce velocemente, creando molte opportunità per gli sviluppatori software. Ci sono già milioni di posti di lavoro nel settore IT (*Information Technology*) in tutto il mondo che supportano applicazioni basate sui big data.

Twitter¹⁹, una fonte primaria di big data

Twitter è una delle fonti di big data preferite dagli sviluppatori. Ogni giorno vengono pubblicati circa 800.000.000 di tweet.¹⁹ Per ciascun tweet, sebbene ci sia un limite di 280 caratteri, sono in effetti circa 10.000 i byte di dati forniti da Twitter ai programmatori interessati all'analisi dei tweet. Quindi, moltiplicando 800.000.000 per 10.000, si ottengono ogni giorno 8.000.000.000.000 byte o 8 terabyte (TB) di dati: si può davvero parlare di "big" data!

Il processo necessario per effettuare previsioni è complicato e spesso oneroso, ma da previsioni accurate possono derivare grandi risultati. Il **data mining** (letteralmente "estrazione di dati") è un processo di ricerca attraverso vaste collezioni di dati, in molti casi big data, per individuare informazioni potenzialmente utili a persone e organizzazioni. Il "sentiment" che si può ricavare dal data mining dei tweet può contribuire a prevedere i risultati delle elezioni, i probabili incassi di un nuovo film e il successo della campagna di marketing di un'azienda. Può inoltre essere d'aiuto per le aziende nell'identificare i punti deboli dell'offerta di prodotti della concorrenza.

19. "Twitter Usage Statistics." Accesso 1 novembre 2020. <https://www.internetlivestats.com/twitter-statistics/>.

✓ Autovalutazione

1. (*Completare*) Un _____ è l'abbreviazione di "binary digit", una cifra che può assumere uno di due valori e che è il più piccolo elemento di informazione in un computer.

Risposta: bit.

2. (*Vero/Falso*) In alcuni sistemi operativi un file è visto semplicemente come una sequenza di byte: qualsiasi struttura dei byte nel file, come la suddivisione in record, è un'elaborazione creata dal programmatore.

Risposta: Vero.

3. (*Completare*) Un database è una raccolta di dati organizzati in maniera tale da permetterne facilmente l'accesso e la manipolazione. Il modello più popolare è il database _____, dove i dati sono immagazzinati in semplici tabelle.

Risposta: relazionale.

1.4 Linguaggi macchina, assembly e di alto livello

I programmatori scrivono le istruzioni in diversi linguaggi di programmazione: alcuni sono comprensibili direttamente dal computer, altri richiedono un passaggio intermedio di traduzione. Oggi si utilizzano centinaia di linguaggi, che possono essere suddivisi in tre tipologie generali:

- linguaggi macchina;
- linguaggi assembly;
- linguaggi di alto livello.

Linguaggi macchina

Qualsiasi computer capisce solamente il proprio **linguaggio macchina**, definito dalla sua struttura hardware. I linguaggi macchina generalmente consistono in stringhe di numeri (codificati alla fine da sequenze di 1 e 0) che istruiscono i computer a eseguire una alla volta le loro operazioni più elementari. Questi linguaggi sono dipendenti dalla macchina (un particolare linguaggio macchina può essere usato solo su quel tipo di computer) e risultano incomprensibili agli esseri umani. Per esempio, quella che segue è una porzione di un programma, scritto in un linguaggio macchina, che somma gli straordinari allo stipendio base e ne memorizza il risultato come stipendio lordo:

```
+1300042774
+1400593419
+1200274027
```

Nel caso pratico che prevede la costruzione di un computer (Esercizi 7.28-7.30), "aprirete" un computer per osservarne la struttura interna. Introdurremo la programmazione in linguaggio macchina e scriverete diversi programmi in questo linguaggio. Per rendere tale esperienza ancor più significativa, costruirete quindi una simulazione software di un computer sul quale potrete eseguire i vostri programmi in linguaggio macchina.

Linguaggi assembly e assembler

Programmare in linguaggio macchina era un'attività troppo lenta e noiosa per molti programmatori. Pertanto, invece di usare stringhe di numeri direttamente comprensibili dal computer, i programmatori iniziarono a usare abbreviazioni derivate dall'inglese per rappresentare le operazioni elementari. Queste abbreviazioni sono diventate la base dei **linguaggi assembly**. Per convertire in linguaggio macchina i programmi scritti in assembly, furono sviluppati programmi di traduzione, detti **assembler** (assemblatori). Anche l'esempio seguente illustra una porzione di un programma scritto in un linguaggio assembly che somma gli straordinari allo stipendio base e ne memorizza il risultato come stipendio lordo:

```
load    stipendioBase
add    straordinari
store  stipendioLordo
```

Sebbene questo codice sia più comprensibile agli esseri umani, non è interpretabile da un computer finché non viene tradotto in linguaggio macchina.

Linguaggi di alto livello e compilatori

Con l'avvento dei linguaggi assembly, l'utilizzo dei computer è aumentato rapidamente, ma i programmatore hanno continuato a usare numerose istruzioni anche per effettuare le operazioni più semplici. Per aumentare la velocità del processo di programmazione sono stati sviluppati **linguaggi di alto livello**, nei quali bastava una singola istruzione per svolgere anche compiti significativi. Un tipico programma scritto in un linguaggio di alto livello contiene diverse istruzioni che costituiscono il **codice sorgente** del programma.

I programmi traduttori chiamati **compilatori** convertono i programmi in linguaggio di alto livello nel linguaggio macchina. I linguaggi di alto livello consentono di scrivere istruzioni che somigliano quasi alla lingua parlata e contengono notazioni matematiche di uso comune. Un programma di tenuta di libri paga scritto in un linguaggio di alto livello potrebbe contenere una singola istruzione come questa:

```
stipendioLordo = stipendioBase + straordinari
```

Dal punto di vista del programmatore, i linguaggi di alto livello sono preferibili ai linguaggi macchina e assembly. Il C è uno dei linguaggi di programmazione di alto livello più utilizzati.

Nel caso pratico che prevede la costruzione di un computer (Esercizi 12.24-12.27), costruirete un compilatore che converte i programmi scritti in linguaggio di alto livello nel Simpletron Machine Language appreso nell'Esercizio 7.28. Gli Esercizi 12.24-12.27 raccolgono l'intero processo di programmazione. Scriverete programmi in un semplice linguaggio di alto livello, userete il compilatore da voi costruito per compilarli e poi li eseguirete sul simulatore Simpletron costruito nell'Esercizio 7.29.

Interpreti

La compilazione di un esteso programma in linguaggio di alto livello nel linguaggio macchina può richiedere al computer una considerevole quantità di tempo. I programmi **interpreti**, sviluppati per eseguire programmi in linguaggi di alto livello direttamente, fanno risparmiare i tempi di compilazione, sebbene lavorino più lentamente dei programmi compilati. Alcuni linguaggi di programmazione, come Java²⁰ e Python²¹, utilizzano una furba strategia mista di compilazione e interpretazione per eseguire i programmi.

✓ Autovalutazione

1. (*Completare*) Per convertire in linguaggio macchina i programmi scritti in assembly, furono sviluppati programmi di traduzione detti _____.

Risposta: assembler.

2. (*Completare*) I programmi _____ sono stati sviluppati per eseguire direttamente programmi scritti in linguaggi di alto livello, evitando così il tempo richiesto dalla compilazione, anche se sono più lenti dei programmi compilati.

Risposta: interpreti.

3. (*Vero/Falso*) I linguaggi di alto livello permettono di scrivere istruzioni che assomigliano alla lingua parlata e che contengono le più comuni notazioni matematiche.

Risposta: Vero.

20. "Java virtual machine." Accesso 2 novembre 2020. https://en.wikipedia.org/wiki/Java_virtual_machine#Bytecode_interpreter_and_just-in-time_compiler.

21. "An introduction to Python bytecode." Accesso 1 novembre 2020. <https://opensource.com/article/18/4/introduction-python-bytecode>.

1.5 Sistemi operativi

I **sistemi operativi** sono sistemi software che rendono l'uso dei computer più comodo per gli utenti, gli sviluppatori di applicazioni e gli amministratori di sistema. Essi forniscono servizi che consentono di eseguire ogni applicazione con sicurezza, in maniera efficiente e concorrentemente con altre applicazioni. Il software che contiene i componenti fondamentali del sistema operativo è chiamato **kernel** (nucleo). Linux, Windows e macOS sono i più diffusi sistemi operativi per computer desktop, e potete utilizzare uno qualsiasi di essi con questo libro. Ciascuno è parzialmente scritto in C. I sistemi operativi per dispositivi mobile più popolari utilizzati su smartphone e tablet sono Android di Google e iOS di Apple.

Windows, un sistema operativo proprietario

Nella metà degli anni Ottanta Microsoft sviluppò il **sistema operativo Windows**, costituito da un'interfaccia grafica costruita sopra il DOS (*Disk Operating System*), un sistema operativo per personal computer estremamente diffuso, con cui gli utenti interagivano digitando comandi. Windows 10 è l'ultimo sistema operativo di Microsoft; include l'assistente personale Cortana per interazioni vocali. Windows è un sistema operativo **proprietario**: è controllato esclusivamente da Microsoft. È di gran lunga il sistema operativo più usato nel mondo.

Linux, un sistema operativo open source

Il **sistema operativo Linux** è tra i più grandi successi del movimento *open source*. Nei primi anni del software è stato dominante il modello proprietario a pagamento. Invece con il concetto di **open source** ("a codice aperto"), individui e aziende contribuiscono a sviluppare, mantenere e migliorare il software, che può essere utilizzato da chiunque per le proprie esigenze, di norma a titolo gratuito, benché soggetto a una gamma di requisiti di licenza (solitamente accomodanti). Il codice open source viene in genere controllato da una platea molto più ampia rispetto al software proprietario, quindi gli errori vengono rimossi più velocemente, rendendo il software più robusto. Il modello open source aumenta la produttività e ha contribuito a una forte innovazione. In questo libro utilizzerete alcuni tra i più noti strumenti e librerie open source.

Tra le tante organizzazioni della comunità open source, elenchiamo di seguito alcune delle più importanti.

- **GitHub** (fornisce strumenti per gestire progetti open source, con milioni di questi in fase di sviluppo).
- **Apache Software Foundation** (originariamente creatori del server web Apache, ora supervisionano più di 350 progetti open source, incluse molte infrastrutture tecnologiche per big data).
- **Eclipse Foundation** (l'ambiente di sviluppo integrato Eclipse aiuta i programmati a sviluppare software facilmente).
- **Mozilla Foundation** (creatori del browser web Firefox).
- **OpenML** (si focalizza su strumenti e dati open source per il machine learning).
- **OpenAI** (fanno ricerca sull'intelligenza artificiale e pubblicano strumenti open source per l'apprendimento con rinforzo, o *reinforcement learning*, usati nella ricerca IA).
- **OpenCV** (si focalizza su strumenti open source di visione artificiale che possono essere usati in vari sistemi operativi e con vari linguaggi di programmazione).
- **Python Software Foundation** (responsabile del linguaggio di programmazione Python).

I rapidi miglioramenti nell'informatica e nelle telecomunicazioni, la diminuzione dei costi e il software open source hanno reso molto più facile e più economico creare oggi un'azienda basata sul software rispetto a dieci anni fa. Facebook, che fu lanciato dalla camera di una residenza universitaria, è stato costruito con un software open source.

Il **kernel di Linux** è il nucleo di molti sistemi operativi open source distribuiti gratuitamente e completi. Sviluppato da una squadra di volontari, è largamente usato nei server, nei personal computer e nei sistemi integrati (come i sistemi alla base di smartphone e smart TV e quelli per automobili). A differenza del codice sorgente di Windows di Microsoft e macOS di Apple, il codice sorgente di Linux è disponibile al pubblico per analisi e modifiche ed è liberamente scaricabile e installabile. Il risultato è che gli utenti di Linux beneficiano

di una numerosissima comunità di sviluppatori che continuamente correggono e migliorano il kernel, e hanno la possibilità di adattare il sistema operativo in base a esigenze specifiche.

macOS e iOS di Apple per dispositivi iPhone® e iPad®

Apple, fondata nel 1976 da Steve Jobs e Steve Wozniak, diventò presto un'azienda leader nel settore dei personal computer. Nel 1979, Jobs e altri dipendenti Apple visitarono lo Xerox PARC (*Palo Alto Research Centre*) per informarsi sul personal computer Xerox che presentava un'interfaccia grafica utente (GUI, *Graphical User Interface*). Quella GUI fu di ispirazione per il primo Apple Macintosh, lanciato nel 1984.

Il linguaggio di programmazione Objective-C, creato nei primi anni Ottanta da Stepstone, aggiunse elementi di programmazione orientata agli oggetti (OOP, *Object-Oriented Programming*) al linguaggio C. Steve Jobs lasciò Apple nel 1985 per fondare la NeXT Inc. Nel 1988 NeXT acquistò la licenza per Objective-C da Stepstone e sviluppò un compilatore e librerie per Objective-C che furono alla base dell'interfaccia utente del sistema operativo NeXTSTEP e di Interface Builder, usato per costruire interfacce grafiche per utenti.

Jobs ritornò in Apple nel 1996, quando NeXT fu acquistata da Apple. Il **sistema operativo macOS** di Apple è un discendente di NeXTSTEP. Apple ha molti altri sistemi operativi proprietari derivati da macOS:

- **iOS** è usato sugli iPhone;
- **iPadOS** è usato sugli iPad;
- **watchOS** è usato sugli Apple Watch;
- **tvOS** è usato sui dispositivi Apple TV.

Nel 2014 Apple presentò il suo nuovo linguaggio di programmazione Swift, diventato open source nel 2015. La comunità di sviluppatori di app Apple è passata in larga parte da Objective-C a Swift. Le app basate su Swift possono importare componenti software Objective-C e C.²²

Android di Google

Android, il sistema operativo con la crescita maggiore nel settore mobile, è basato sul kernel di Linux, sul linguaggio di programmazione Java e, ora, sul linguaggio di programmazione open source Kotlin. Android è gratuito e open source. Sebbene non sia possibile sviluppare app Android esclusivamente in C, è possibile incorporare il codice C nelle app Android.²³

Secondo idc.com, nel 2020 Android possedeva l'84,8% delle quote globali del mercato smartphone, contro il 15,2% di Apple.²⁴ Il sistema operativo Android è usato in numerosi smartphone, dispositivi di lettura elettronica, tablet, TV, schermi touch nei negozi, automobili, robot, lettori multimediali e molto altro.

Miliardi di dispositivi computerizzati

Oggi vengono usati miliardi di personal computer e un numero ancora più grande di dispositivi mobile. La crescita esplosiva di smartphone, tablet e altri dispositivi sta creando notevoli opportunità per la programmazione di applicazioni mobile. La tabella seguente elenca molti dispositivi computerizzati, ognuno dei quali può far parte dell'Internet delle cose (vedi Paragrafo 1.11).

22. "Imported C and Objective-C APIs." Accesso 3 novembre 2020. https://developer.apple.com/documentation/swift/imported_c_and_objective-c_apis.

23. "Add C and C++ code to your project." Accesso 3 novembre 2020. <https://developer.android.com/studio/projects/add-native-code>.

24. "Smartphone Market Share." Accesso 24 dicembre 2020. <https://www.idc.com/promo/smartphone-market-share/os>.

Dispositivi computerizzati		
Automobili	e-reader	Sistemi di sicurezza domestica
Carte di credito	Macchine per tomografia	Sistemi per lotterie
Cellulari	Parchimetri	Smartcard
Computer desktop	Personal computer	Smartphone
Console per videogiochi	Riproduttori blu-ray	Sportelli POS
Contatori logici	Risonanze magnetiche	Stampanti
Controllori logici	Robot	Tablet
Decoder	Sensori ottici	Televisioni
Decoder per TV digitale	Server	Terminali di vendita
Dispositivi medici	Sistemi di diagnosi per automobili	Termostati
Elettrodomestici	Sistemi di navigazione GPS	Tessere d'accesso

✓ Autovalutazione

1. (*Completare*) Windows è un sistema operativo _____, ovvero controllato in esclusiva da Microsoft.
Risposta: proprietario.
2. (*Vero/Falso*) Il codice proprietario spesso viene controllato da una platea molto più ampia rispetto al software open source, quindi gli errori vengono rimossi più velocemente.
Risposta: Falso. Il codice open source viene in genere controllato da una platea molto più ampia rispetto al software proprietario, quindi gli errori vengono rimossi più velocemente.
3. (*Vero/Falso*) iOS domina il mercato globale degli smartphone rispetto ad Android.
Risposta: Falso. Android nel 2020 controllava l'84,8% del mercato smartphone, ma le app iOS guadagnano quasi il doppio delle app Android.²⁵

1.6 Il linguaggio di programmazione C

Il linguaggio C è l'evoluzione di due precedenti linguaggi, BCPL²⁶ e B²⁷. BCPL è stato sviluppato nel 1967 da Martin Richards come linguaggio per scrivere sistemi operativi e compilatori. Ken Thompson ha ripreso molte caratteristiche del suo linguaggio B da quelle omologhe del BCPL e nel 1970 ha usato B per creare versioni iniziali del sistema operativo UNIX presso i Laboratori Bell.

Il linguaggio C si è evoluto da B per opera di Dennis Ritchie presso i Laboratori Bell ed è stato originalmente implementato nel 1972.²⁸ C è da sempre comunemente conosciuto come il linguaggio di sviluppo del sistema operativo UNIX. Molti dei principali sistemi operativi odierni sono scritti in C e/o C++. Questo linguaggio è generalmente indipendente dall'hardware. Con una progettazione accurata è possibile scrivere programmi nel linguaggio C **portabili** sulla maggior parte dei computer.

Ideato per le prestazioni

Il linguaggio C è largamente usato per sviluppare sistemi che richiedono prestazioni elevate, come i sistemi operativi, i sistemi embedded, i sistemi in tempo reale e i sistemi di telecomunicazione.

25. "Global App Revenue Reached \$50 Billion in the First Half of 2020, Up 23% Year-Over-Year." Accesso 1 novembre 2020. <https://sensortower.com/blog/app-revenue-and-downloads-1h-2020>.

26. "BCPL." Accesso 1 novembre 2020. <https://en.wikipedia.org/wiki/BCPL>.

27. "B (programming language)." Accesso 1 novembre 2020. [https://en.wikipedia.org/wiki/B_\(programming_language\)](https://en.wikipedia.org/wiki/B_(programming_language)).

28. "C (programming language)." Accesso 1 novembre 2020. [https://en.wikipedia.org/wiki/C_\(programming_language\)](https://en.wikipedia.org/wiki/C_(programming_language)).

Applicazione	Descrizione
Sistemi operativi	La portabilità e le prestazioni del linguaggio C lo rendono adatto a implementare sistemi operativi, come Linux, parti di Windows di Microsoft e Android di Google. macOS di Apple è costruito con Objective-C, derivato dal linguaggio C. Parleremo di alcuni importanti sistemi operativi per desktop/notebook e per dispositivi mobili nel Paragrafo 1.5.
Sistemi embedded	La grande maggioranza dei microprocessori prodotti ogni anno è inserita in dispositivi diversi dai computer di tipo general purpose. Questi sistemi embedded comprendono sistemi di navigazione, elettrodomestici intelligenti, sistemi di sicurezza privata, smartphone, robot, incroci intelligenti per il traffico e altro. Il linguaggio C è uno dei più popolari linguaggi di programmazione per lo sviluppo di sistemi embedded, che solitamente necessitano di operare il più velocemente possibile e di risparmiare memoria. Per esempio, il sistema frenante ABS (<i>Anti Block System</i>) di un'auto deve rispondere immediatamente per far rallentare o fermare il veicolo senza slittamento; le periferiche dei videogiochi devono rispondere istantaneamente per evitare qualsiasi sfasamento tra la periferica e l'azione nel gioco.
Sistemi in tempo reale	I sistemi in tempo reale sono spesso utilizzati per applicazioni in missioni critiche che richiedono tempi di risposta quasi istantanei e prevedibili. I sistemi in tempo reale richiedono un lavoro continuo. Per esempio, un sistema di controllo del traffico aereo deve costantemente monitorare la posizione e la velocità degli aerei e riportare subito tali informazioni ai controllori di volo, in modo che questi possano avvisare gli aerei per far loro cambiare rotta nel caso di una possibile collisione.
Sistemi di telecomunicazione	I sistemi di telecomunicazione hanno necessità di instradare verso le loro destinazioni grandi quantità di dati velocemente, per assicurare che le informazioni audio e video siano trasmesse bene e senza ritardi.

Alla fine degli anni Settanta il linguaggio C si è evoluto in quello che ora è chiamato “C tradizionale”. La pubblicazione nel 1978 del libro di Kernighan e Ritchie *Il linguaggio di programmazione C* attirò una grande attenzione sul linguaggio. Il libro è diventato uno dei testi di informatica di maggior successo di tutti i tempi.

Standardizzazione

La rapida espansione del linguaggio C su varie **piattaforme hardware** (ovvero, tipi di hardware per computer) ha portato a molte sue versioni simili tra loro ma spesso incompatibili. Questo ha costituito un problema serio per i programmatore che avevano bisogno di sviluppare un codice che girasse su diverse piattaforme. Divenne chiaro che si rendeva necessaria una versione standard del linguaggio. Nel 1983, all'interno dell'American National Standards Committee per i computer e l'elaborazione delle informazioni (X3), fu creato il comitato tecnico X3J11, con l'obiettivo di “fornire una definizione del linguaggio non ambigua e indipendente dalla macchina”. Nel 1989 lo standard fu approvato negli Stati Uniti tramite l'**American National Standards Institute (ANSI)**, poi in tutto il mondo tramite l'**International Standards Organization (ISO)**. Questa versione fu chiamata semplicemente linguaggio C standard.

I linguaggi C11 e C18 standard

Analizziamo l'ultimo standard del C (chiamato C11), che è stato approvato nel 2011 e aggiornato con correzioni di bug nel 2018 (chiamato C18). Il C11 rifinisce ed espande le capacità del linguaggio C. Abbiamo integrato in specifici paragrafi del testo e nell'Appendice C (in sezioni facili da includere o omettere) molte delle nuove funzionalità implementate nei principali compilatori C. L'attuale documento del C standard è indicato come *ISO/IEC 9899:2018*. Le copie possono essere ordinate da

<https://www.iso.org/standard/74528.html>

Secondo il comitato che se ne occupa, è probabile che il prossimo standard del C venga rilasciato nel 2022.²⁹

29. “Programming Language C — C2x Charter.” Accesso 4 novembre 2020. <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2086.htm>.

Dal momento che il C è un linguaggio indipendente dall'hardware e largamente diffuso, le applicazioni scritte in C possono spesso essere eseguite con poche modifiche, o nessuna del tutto, su un'ampia gamma di sistemi di computer.

✓ Autovalutazione

1. (*Completare*) Il linguaggio C è l'evoluzione di due precedenti linguaggi, _____ e _____.

Risposta: BCPL, B.

2. (*Vero/Falso*) È possibile scrivere programmi in C che siano portabili sulla maggior parte dei computer.

Risposta: Vero.

1.7 Libreria Standard del C e librerie open source

I programmi in C sono costituiti da moduli chiamati **funzioni**. È possibile programmare *ex novo* tutte le funzioni necessarie per un programma in C, ma la maggior parte dei programmatori in C si avvale della ricca collezione di funzioni esistenti, chiamata **Libreria Standard del C**. Sono quindi necessarie due cose per imparare a programmare in C:

- imparare il linguaggio C stesso, e
- imparare a usare le funzioni della Libreria Standard del C.

In tutto il libro saranno esaminate molte di queste funzioni. Ai programmatori che hanno necessità di comprendere a fondo le funzioni della libreria, il modo in cui vanno implementate e utilizzate per scrivere un codice portatile, consigliamo vivamente la lettura del libro *The Standard C Library* di P. J. Plauger. In questo testo useremo e spiegheremo molte funzioni della libreria del C.

Il testo incoraggia un **approccio a blocchi** per la creazione di programmi. Nella programmazione in C si usano abitualmente i seguenti blocchi:

- funzioni della Libreria Standard del C;
- funzioni delle librerie open source in C;
- funzioni che create voi stessi;
- funzioni che altri (di cui vi fidate) hanno creato e reso disponibili per voi.

Il vantaggio di creare le proprie funzioni sta nel sapere esattamente come queste opereranno. Lo svantaggio consiste nel fatto che il lavoro di progettazione, sviluppo, messa a punto e regolazione delle prestazioni di nuove funzioni richiede una considerevole quantità di tempo. In questo libro puntiamo a utilizzare le funzioni della Libreria Standard del C per sostenere i vostri sforzi nello sviluppo di programmi, evitando di “reinventare la ruota” (**riusabilità del software**).

Dal momento che le funzioni della Libreria Standard del C sono scritte con cura per offrire prestazioni efficienti, è opportuno utilizzarle, anziché scriverne di nuove, per avere maggiori probabilità di migliorare le prestazioni e la portabilità del programma.

Librerie open source

Esistono parecchie librerie di terze parti e open source in C che vi permettono di svolgere attività significative con una quantità di codice relativamente modesta. Potete per esempio trovare più di 32.000 repository nella categoria C di GitHub:

<https://github.com/topics/c>

Inoltre, ci sono pagine come *Awesome C*

<https://github.com/kozross/awesome-c>

che forniscono elenchi revisionati di librerie in C utili in molte diverse aree di applicazione.

✓ Autovalutazione

1. (*Completare*) La maggior parte dei programmati in C sfrutta la ricca raccolta di funzioni esistenti chiamata _____.

Risposta: Libreria Standard del C.

2. (*Completare*) L'utilizzo di funzioni già esistenti, evitando di "reinventare la ruota", viene detto _____.

Risposta: riusabilità del software.

1.8 Altri linguaggi di programmazione popolari

L'elenco seguente è una breve introduzione ad altri popolari linguaggi di programmazione:

- *BASIC* è stato sviluppato negli anni Sessanta del secolo scorso al Dartmouth College per aiutare i principianti a familiarizzare con le tecniche di programmazione. Molte delle sue ultime versioni sono orientate agli oggetti.
- *C++*, che è basato sul C, è stato sviluppato da Bjarne Stroustrup nei primi anni Ottanta del secolo scorso ai Bell Laboratories. C++ potenzia il C aggiungendo anche funzionalità per la programmazione a oggetti. Introdurremo la programmazione orientata agli oggetti nell'Appendice D.
- *Python*, un linguaggio orientato agli oggetti, è stato rilasciato pubblicamente nel 1991. Sviluppato da Guido van Rossum del National Research Institute for Mathematics and Computer Science di Amsterdam, Python è diventato rapidamente uno dei linguaggi di programmazione più popolari al mondo, in particolare per l'informatica didattica e scientifica, e nel 2017 ha superato il linguaggio di programmazione R come il più popolare linguaggio di programmazione per la scienza dei dati (*data science*).^{30,31,32} Ecco alcuni motivi per i quali Python è popolare:^{33,34,35} è open source, gratuito e ampiamente disponibile; è supportato da un'enorme comunità open source; è relativamente facile da imparare; il suo codice è più facile da leggere rispetto a quello di molti altri linguaggi di programmazione popolari; migliora la produttività degli sviluppatori con ampie librerie standard e migliaia di librerie open source di terze parti; è popolare nello sviluppo web e nell'intelligenza artificiale, che sta godendo di una crescita esplosiva, in parte per via del suo rapporto speciale con la scienza dei dati; è ampiamente utilizzato nella comunità finanziaria.³⁶
- *Java*: nel 1991 Sun Microsystems finanziò un progetto di ricerca interno guidato da James Gosling che produsse questo linguaggio di programmazione a oggetti basato sul C++. Dare la possibilità agli sviluppatori di scrivere programmi eseguibili su un gran numero di sistemi era uno degli obiettivi chiave di questo linguaggio. Java è usato per sviluppare applicazioni aziendali, migliorare le funzionalità dei server web (i computer che forniscono il contenuto ai browser), fornire applicazioni sui dispositivi dei consumatori (smartphone, tablet, decoder per televisioni, elettrodomestici, automobili ecc.) e per molti altri scopi. Java originariamente era il linguaggio chiave per lo sviluppo delle app di Android, mentre ora vengono utilizzati anche altri linguaggi.

30. "5 things to watch in Python in 2017." Accesso 1 novembre 2020. <https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017>.

31. "Python overtakes R, becomes the leader in Data Science, Machine Learning platforms," Accesso 1 novembre 2020. <https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html>.

32. "Data Science Job Report 2017: R Passes SAS, But Python Leaves Them Both Behind." Accesso 1 novembre 2020. <https://www.r-bloggers.com/data-science-job-report-2017-r-passes-sas-but-python-leaves-them-both-behind/>.

33. "Why Learn Python? Here Are 8 Data-Driven Reasons." Accesso 1 novembre 2020. <https://dbader.org/blog/why-learn-python>.

34. "Why Learn Python." Accesso 1 novembre 2020. <https://simpleprogrammer.com/7-reasons-why-you-should-learn-python>.

35. "5 things to watch in Python in 2017." Accesso 1 novembre 2020. <https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017>.

36. Kolanovic, M., Krishnamachari, R., *Big Data and AI Strategies: Machine Learning and Alternative Data Approach to Investing* (J.P. Morgan, 2017).

- *C#* (basato sul *C++* e su *Java*) è uno dei tre principali linguaggi orientati agli oggetti di Microsoft; gli altri due sono *Visual C++* e *Visual Basic*. *C#* fu sviluppato per l'integrazione del Web nei programmi per computer e ora è largamente utilizzato per sviluppare le più svariate applicazioni. Come parte integrante della strategia open source di Microsoft, anche alcune versioni di *C#* e *Visual Basic* sono ora open source.
- *JavaScript* è un linguaggio di scripting molto comune, principalmente usato per aggiungere funzionalità alle pagine web, come per esempio animazioni o l'interazione con l'utente. La maggior parte dei browser lo supporta. Molte librerie di visualizzazione per *Python* producono codice *JavaScript*, così da poterci interagire nel proprio browser web. Anche strumenti come *NodeJS* permettono di utilizzare *JavaScript* fuori dai browser.
- *Swift*, introdotto nel 2014, è il linguaggio di programmazione di Apple per la produzione di app per *iOS* e *macOS*. *Swift* è un linguaggio moderno che include le caratteristiche più popolari di linguaggi come *Objective-C*, *Java*, *C#*, *Ruby*, *Python* e altri ancora. *Swift* è open source, quindi può essere usato anche su piattaforme diverse da quelle Apple.
- *R* è un linguaggio di programmazione open source molto popolare per l'analisi statistica dei dati. *Python* e *R* sono i due linguaggi più usati nel mondo della data science.

✓ Autovalutazione

1. (*Completere*) Oggigiorno, il codice di molti sistemi operativi multiuso e di sistemi in cui le prestazioni sono critiche è scritto in _____.

Risposta: C o C++.

2. (*Completere*) Dare la possibilità agli sviluppatori di scrivere programmi eseguibili su un gran numero di sistemi è uno degli obiettivi chiave di _____.

Risposta: Java.

3. (*Vero/Falso*) R è il linguaggio di programmazione per l'analisi statistica dei dati più popolare.

Risposta: Falso. Nel 2017, *Python* ha superato *R* come il più popolare linguaggio di programmazione per la scienza dei dati.

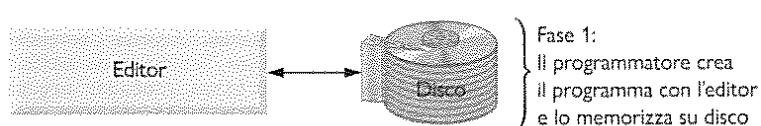
1.9 Un tipico ambiente di sviluppo per la programmazione in C

I sistemi di supporto alla programmazione in C consistono generalmente di diverse parti: un ambiente di sviluppo, il linguaggio e la Libreria Standard del C. L'analisi che segue illustra il tipico ambiente di sviluppo per il linguaggio C.

Per essere eseguiti, i programmi in C passano normalmente attraverso sei fasi: **editing** (redazione), **pree-laborazione**, **compilazione**, **linking** (collegamento), **loading** (caricamento) ed **esecuzione**. Sebbene questo sia un libro di testo generico sul C (scritto senza tener conto dei dettagli specifici di un particolare sistema operativo), in questo paragrafo ci concentreremo su un tipico sistema per il linguaggio C basato su Linux. Nel Paragrafo 1.10 proverete a creare ed eseguire programmi C su Windows, macOS e/o Linux.

1.9.1 Fase 1: creazione di un programma

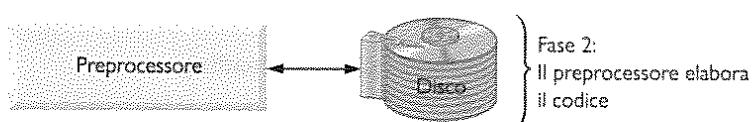
La Fase 1 (nel diagramma seguente) consiste nell'**editing** di un file per mezzo di un **programma editor**:



Due editor molto diffusi sui sistemi Linux sono Vi ed Emacs. Gli ambienti di sviluppo integrato (IDE) per C e C++ come Visual Studio di Microsoft e Xcode di Apple forniscono editor integrati. Si scrive un programma in C con l'editor, si apportano correzioni se necessario, poi si memorizza il programma su un dispositivo di memoria secondaria, come un disco rigido. I nomi dei file che contengono programmi in C devono terminare con l'estensione .c.

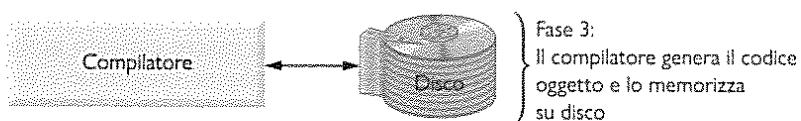
1.9.2 Fasi 2 e 3: preelaborazione e compilazione di un programma in C

Nella Fase 2 (mostrata nel diagramma seguente) si dà il comando di **compilazione** del programma:



Il compilatore traduce il programma in C in un codice nel linguaggio macchina (denominato anche **codice oggetto**). In un sistema di sviluppo per il C viene eseguito automaticamente un programma di **preelaborazione** prima che inizi la fase di traduzione da parte del compilatore. Il **preprocessore C** obbedisce a speciali comandi chiamati **direttive per il preprocessore**, che eseguono manipolazioni di testo su file con codice sorgente di un programma. Queste manipolazioni consistono nell'inserire il contenuto di altri file e in varie sostituzioni di testo. Le più comuni direttive per il preprocessore sono esaminate nei primi capitoli; nel Capitolo 14 sono discusse altre funzionalità del preprocessore.

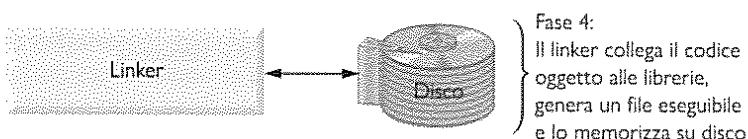
Nella Fase 3 (mostrata nel diagramma seguente), il compilatore traduce il programma in C nel codice in linguaggio macchina:



Quando il compilatore non riesce a riconoscere un'istruzione perché questa viola le regole del linguaggio, si verifica un **errore di sintassi**. Per aiutare a individuare e a correggere l'istruzione scorretta, il compilatore dà un messaggio di errore. Lo Standard del C non specifica la forma esatta per i messaggi di errore dati dal compilatore, così i messaggi di errore che si vedono sul proprio sistema possono differire da quelli che si vedono su altri sistemi. Gli errori di sintassi sono chiamati anche **errori di compilazione** o **errori in fase di compilazione**.

1.9.3 Fase 4: linking

La fase successiva (mostrata nel diagramma seguente) è chiamata **linking**:



I programmi in C usano di solito funzioni definite altrove, come nelle librerie standard, nelle librerie open source o nelle librerie private di un particolare progetto. Il codice oggetto prodotto dal compilatore C contiene solitamente "buchi" dovuti a queste parti mancanti. Un **linker** collega il codice oggetto di un programma con il

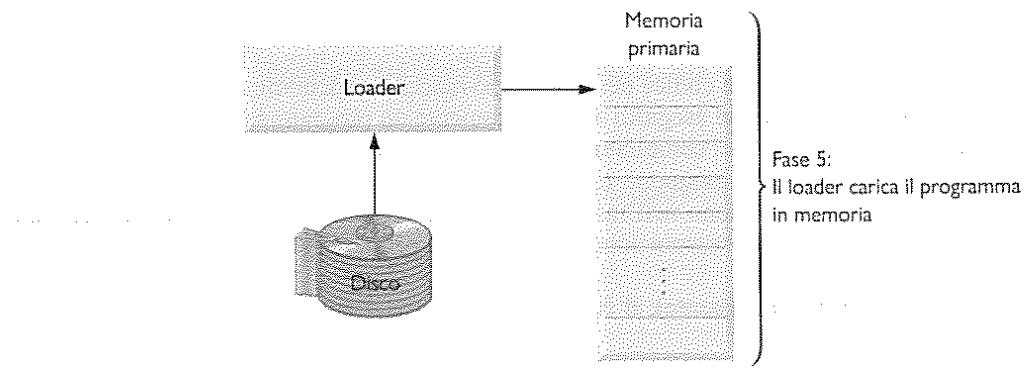
codice delle funzioni mancanti, così da produrre un'**immagine eseguibile** (senza pezzi mancanti). Su un tipico sistema Linux, il comando per la compilazione e il linking di un programma è chiamato **gcc** (il compilatore GNU C). Per compilare ed effettuare il link di un programma chiamato `welcome.c` usando l'ultimo standard del C (C18), scrivete

```
gcc -std=c18 welcome.c
```

al prompt di Linux e premete il tasto *Invio*. I comandi di Linux sono sensibili all'uso del carattere minuscolo/maiuscolo. Se la compilazione e il linking del programma sono effettuati correttamente, viene prodotto (per default) un file chiamato `a.out`, che è l'immagine eseguibile di `welcome.c`.

1.9.4 Fase 5: loading

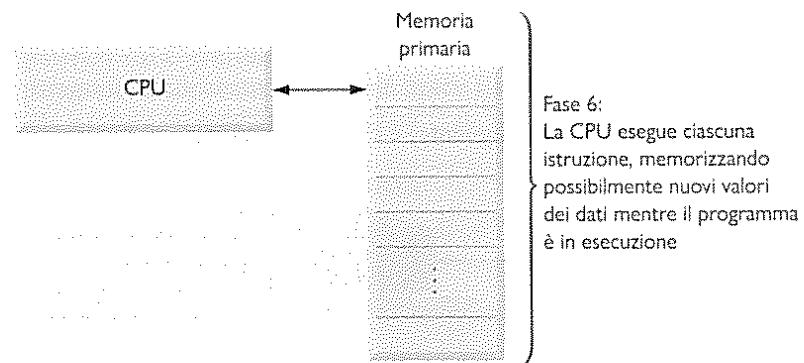
La fase successiva (mostrata nel diagramma seguente) è chiamata **loading**:



Prima che un programma possa essere eseguito, deve innanzitutto essere caricato nella memoria centrale e questo si fa per mezzo del **loader**, il quale prende dal disco l'immagine eseguibile e la trasferisce nella memoria primaria. Vengono anche caricati componenti aggiuntivi presi da librerie condivise che supportano il programma.

1.9.5 Fase 6: esecuzione

Alla fine, nell'ultima fase (mostrata nel diagramma seguente), il computer, sotto il controllo della sua CPU, **esegue** il programma un'istruzione per volta:



Per caricare ed eseguire il programma su un sistema Linux, scrivete `./a.out` al prompt di Linux e premete *Invio*.

1.9.6 Problemi che possono presentarsi al momento dell'esecuzione

I programmi non funzionano sempre al primo tentativo. Ognuna delle fasi precedenti può fallire per via di vari errori che analizzeremo. Per esempio, se un programma in esecuzione tentasse una divisione per zero (operazione non ammessa sui computer, proprio come in aritmetica), apparirebbe un messaggio di errore; a quel punto dovreste ritornare alla fase di editing, apportare le necessarie correzioni e procedere di nuovo attraverso le successive fasi per verificare che le modifiche funzionino correttamente.

 Errori come una divisione per zero che si verificano quando i programmi sono in esecuzione vengono chiamati errori in fase di esecuzione. Dividere per zero è in genere un errore irreversibile, uno di quelli che causano immediatamente la terminazione del programma, senza che questo abbia eseguito con successo il proprio lavoro. Gli errori non irreversibili permettono al programma di essere eseguito fino alla fine, producendo però spesso risultati scorretti.

1.9.7 Gli stream standard input, standard output e standard error

La maggior parte dei programmi in C riceve in ingresso e/o produce in uscita dei dati. Certe funzioni del C ricevono il loro input da **stdin** (**stream standard input**, il flusso standard di dati in entrata), che è normalmente la tastiera. I dati sono spesso inviati a **stdout** (**stream standard output**, il flusso standard di dati in uscita), che è normalmente lo schermo del computer. Quando diciamo che un programma stampa un risultato, normalmente intendiamo che il risultato è mostrato sullo schermo. I dati possono essere inviati a dispositivi quali dischi e stampanti. Vi è anche uno **stream standard error** (flusso standard di dati per gli errori) denominato **stderr**, che normalmente è connesso allo schermo e usato per mostrare messaggi di errore. È uso comune inviare i dati regolari in uscita (**stdout**) a un dispositivo diverso dallo schermo mantenendo **stderr** collegato allo schermo in modo che l'utente possa essere immediatamente informato degli errori.

✓ Autovalutazione

1. *(Completare)* Per essere eseguiti, i programmi in C passano normalmente attraverso sei fasi: _____, _____, _____, _____, _____ e _____.

Risposta: editing, preelaborazione, compilazione, linking, loading, esecuzione.

2. *(Completare)* Quando il compilatore non riesce a riconoscere un'istruzione perché questa viola le regole del linguaggio, si verifica un _____.

Risposta: errore di sintassi.

3. *(Completare)* Errori che si verificano quando i programmi sono in esecuzione vengono chiamati _____.

Risposta: errori in fase di esecuzione.

1.10 Test: eseguire un'applicazione in C negli ambienti Windows, Linux e macOS

In questo paragrafo compileremo, eseguiremo e interagiremo con la nostra prima applicazione in C. Inizieremo con un gioco che consiste nell'indovinare un numero scelto a caso fra 1 e 1.000. Se il numero viene indovinato, il gioco finisce; se non viene indovinato, l'applicazione indica se è maggiore o minore del numero corretto. Non c'è limite al numero di risposte che si possono dare, ma si deve riuscire a indovinare uno dei numeri di questa serie entro un massimo di dieci tentativi. Dietro questo gioco vi è una raffinata tecnica informatica (in uno dei prossimi capitoli esploreremo la tecnica della ricerca binaria).

Creerete questa applicazione negli esercizi del Capitolo 5. Di solito, questa applicazione seleziona a caso le risposte corrette. Abbiamo disabilitato la selezione casuale per i test guidati. L'applicazione utilizza la stessa risposta corretta ogni volta che viene eseguita. In questo modo, si possono utilizzare le stesse risposte che proponiamo qui e vedere gli stessi risultati. Questa risposta può variare a seconda del compilatore.

Riepilogo dei test

Vedremo come creare un'applicazione in C utilizzando:

- Microsoft Visual Studio 2019 Community Edition per Windows (Paragrafo 1.10.1);
- Clang in Xcode su macOS (Paragrafo 1.10.2);
- GNU gcc in una shell su Linux (Paragrafo 1.10.3);
- GNU gcc in una shell che viene eseguita all'interno del contenitore Docker della GNU Compiler Collection (GCC) (Paragrafo 1.10.4).

Leggete pure soltanto il paragrafo corrispondente al vostro setup.

Sono disponibili molti ambienti di sviluppo in cui è possibile compilare, costruire ed eseguire applicazioni in C. Se lo strumento utilizzato nel vostro corso non compare tra quelli qui trattati, chiedete informazioni in merito al vostro insegnante.

1.10.1 Compilare ed eseguire un'applicazione in C con Visual Studio 2019 Community Edition su Windows 10

In questo paragrafo viene mostrato come eseguire un programma in C su Windows utilizzando Microsoft Visual Studio 2019 Community Edition. Sono disponibili numerose versioni di Visual Studio, che potrebbero essere leggermente differenti da quella qui presentata per quanto riguarda opzioni, menu e istruzioni. Quindi da ora in avanti ci riferiremo semplicemente a “Visual Studio” o “l’IDE”.

Fase 1: controllare il setup

Leggete la sezione “Prima di cominciare”, se non l'avete già fatto, relativa alle istruzioni per installare l'IDE e scaricare gli esempi di codice del libro.

Fase 2: lanciare Visual Studio

Lanciate Visual Studio dal menu **Start**. Premete il tasto *Esc* per uscire dalla finestra iniziale di Visual Studio. Non fate clic sulla **X** nell'angolo in alto a destra, che farebbe chiudere Visual Studio. Potete accedere a questa finestra in ogni momento selezionando **File > Start Window**. Usiamo il simbolo **>** per indicare la selezione di una voce da un menu, quindi per esempio **File > Open** significa “selezionate la voce di menu **Open** dal menu **File**”.

Fase 3: creare un progetto

Un **progetto** consiste in un gruppo di file correlati, come per esempio i file di codice sorgente in C che compongono un'applicazione. Visual Studio organizza le applicazioni in progetti e **soluzioni**, le quali contengono uno o più progetti. Per creare applicazioni di larga scala, i programmati si servono di soluzioni multiprogetto, mentre i nostri esempi richiedono solo soluzioni per un singolo progetto. Inizierete con un progetto vuoto (**Empty Project**) a cui aggiungerete alcuni file. Per creare un progetto:

1. Selezionate **File > New > Project...** per visualizzare la finestra di dialogo **Create a new project**.
2. Selezionate il modello (*template*) **Empty Project** con i tag **C++, Windows** e **Console**. Anche se Visual Studio non ha un compilatore C, il suo compilatore Visual C++ è in grado di compilare la maggior parte dei programmi in C. Il modello da noi utilizzato è adatto per programmi che vengono eseguiti dalla riga di comando in una finestra del prompt dei comandi. Possono esserci molti altri modelli di progetto, a seconda della versione e delle opzioni di Visual Studio che avete installato. Potete filtrare le scelte tramite la casella **Search for templates** con i relativi elenchi a discesa. Fate clic su **Next** per visualizzare la finestra di dialogo **Configure your new project**.
3. Specificate il nome e la posizione del progetto (**Project name** e **Location**). Come nome del progetto abbiamo indicato **c_test** e come posizione la cartella **examples** di questo libro, che assumiamo si trovi nella cartella **Documents** del vostro account utente. Fate clic su **Create** per aprire il vostro nuovo progetto in Visual Studio.

A questo punto, Visual Studio crea il nuovo progetto inserendolo nella cartella

`C:\Users\VostroAccountUtente\Documents\examples`

(o in un'altra cartella da voi indicata) e apre la finestra principale di Visual Studio.

Quando fate l'editing del codice in C, Visual Studio visualizza ogni file in una scheda separata nella finestra. La finestra **Solution Explorer**, agganciata a sinistra o a destra della finestra principale di Visual Studio, serve per vedere e gestire i file della vostra applicazione. Negli esempi di questo libro, memorizzerete i file di codice sorgente di ciascun programma nella cartella **Source Files**. Per visualizzare **Solution Explorer** selezionate **View > Solution Explorer**.

Fase 4: aggiungere il file `GuessNumber.c` nel progetto

Aggiungiamo ora il file `GuessNumber.c` nel progetto. Nella finestra **Solution Explorer**:

1. Fate clic con il tasto destro del mouse sulla cartella **Source Files** e selezionate **Add > Existing Item...**
2. Navigate nella finestra di dialogo che appare fino alla sottocartella `ch01` della cartella `examples` del libro, selezionate `GuessNumber.c` e fate clic su **Add**.³⁷

Fase 5: configurare la versione del compilatore per il vostro progetto e disabilitare un messaggio di errore di Microsoft

Come accennato nella sezione “Prima di cominciare”, Visual C++ è in grado di compilare la maggior parte dei programmi in C. Il compilatore Visual C++ supporta molte versioni dello standard C++. Useremo il compilatore C++17 di Microsoft, che dobbiamo configurare nelle impostazioni del nostro progetto:

1. Fate clic con il tasto destro del mouse sul nodo del progetto `c_test` nella finestra **Solution Explorer** e selezionate **Properties** per visualizzare la finestra di dialogo **C_test Property Pages** del progetto.
2. Nell'elenco a discesa **Configuration**, modificate **Active(Debug)** in **All Configurations**. Nell'elenco a discesa **Platform**, modificate **Active(Win32)** in **All Platforms**.
3. Nella colonna sinistra, espandete il nodo **C/C++**, quindi selezionate **Language**.
4. Nella colonna destra, fate clic sul campo alla destra di **C++ Language Standard**, fate clic sulla freccia che punta verso il basso e selezionate **ISO C++17 Standard (/std:c++17)**.³⁸
5. Nella colonna sinistra, nel nodo **C/C++**, selezionate **Preprocessor**.
6. Nella colonna destra, alla fine del valore per **Preprocessor Definitions**, inserite
`;_CRT_SECURE_NO_WARNINGS`
7. Nella colonna sinistra, nel nodo **C/C++**, selezionate **General**.
8. Nella colonna destra, fate clic nel campo alla destra di **SDL checks**, fate clic sulla freccia che punta verso il basso e selezionate **No (/sdl-)**.
9. Fate clic su **OK** per salvare le modifiche effettuate.

Le operazioni ai punti 6 e 8 eliminano gli avvisi e i messaggi di errore generati dal Visual C++ di Microsoft per diverse funzioni della libreria del C che useremo nel corso del libro. L'argomento verrà approfondito nel Paragrafo 3.13.

37. Nel caso di programmi con più file di codice sorgente che vedrete nei capitoli successivi, selezionate tutti i file per il programma considerato. Quando comincerete a creare programmi per conto vostro, potrete fare clic con il tasto destro del mouse sulla cartella **Source Files** e selezionare **Add > New Item...** per visualizzare una finestra di dialogo che vi permetterà di aggiungere un nuovo file. Dovrete inoltre modificare l'estensione dei nomi di file per i vostri programmi in C da `.cpp` a `.c`.

38. Al momento della stesura di questo libro, Microsoft era ancora in fase di completamento del supporto per lo standard C++20. Una volta disponibile, dovreste usare la versione **ISO C++20 Standard (/std:c++20)**.

Fase 6: compilare ed eseguire il progetto

Procediamo ora a compilare ed eseguire il progetto, così potrete testare l'applicazione. Selezionate **Debug > Start without debugging** oppure premete *Ctrl + F5*. Se il programma viene compilato correttamente, Visual Studio apre una finestra del prompt dei comandi ed esegue il programma. Per una maggiore leggibilità, abbiamo modificato lo schema dei colori³⁹ e la dimensione dei caratteri del prompt dei comandi:

```
C:\Users\PaulDeitel\Documents\examples\cpp20\test\Debug\c_\test.exe
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? =
```

Fase 7: inserire la prima risposta

Al prompt ?, digitate **500** e premete *Invio*. L'applicazione visualizza "Too high. Try again." per indicare che il valore inserito è maggiore del numero scelto dall'applicazione come risposta esatta:

```
C:\Users\PaulDeitel\Documents\examples\cpp20\test\Debug\c_\test.exe
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? =
```

Fase 8: inserire un'altra risposta

Al prompt successivo, digitate **250** e premete *Invio*. L'applicazione visualizza "Too high. Try again." per indicare che il valore inserito è maggiore del numero scelto dall'applicazione come risposta esatta:

```
C:\Users\PaulDeitel\Documents\examples\cpp20\test\Debug\c_\test.exe
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too high. Try again.
? =
```

Fase 9: inserire di ulteriori risposte

Continuate a giocare introducendo valori finché non indovinate il numero esatto. A quel punto l'applicazione mostrerà "Excellent! You guessed the number!"

39. Se desiderate modificare i colori del prompt dei comandi sul vostro sistema, fate clic con il tasto destro del mouse sulla barra del titolo e selezionate **Properties**. Nella finestra di dialogo "**Command Prompt**" **Properties**, fate clic sulla scheda **Colors** e scegliete i vostri colori preferiti per testo e sfondo.

```
C:\Users\PaulDette\Documents\examples\rpp39_testDebugC.exe
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too high. Try again.
? 125
Too high. Try again.
? 63
Too high. Try again.
? 31
Too low. Try again.
? 47
Too high. Try again.
? 39
Too low. Try again.
? 43
Too high. Try again.
? 41
Too low. Try again.
? 42

Excellent! You guessed the number!
Would you like to play again?
Please type (1=yes, 2=no)? =
```

Fase 10: continuare il gioco o uscire dall'applicazione

Dopo che è stato indovinato il numero corretto, l'applicazione domanda se volete continuare a giocare. Al prompt “Please type (1=yes, 2=no) ?”, inserite **1** per giocare ancora, con un nuovo numero da indovinare, oppure inserite **2** per terminare l'applicazione. Ogni volta che eseguirete questa applicazione (*Fase 6*), vi verranno proposti gli stessi numeri da indovinare. Per giocare con numeri a caso, usate la versione di `GuessNumber.c` che troverete nella sottocartella `randomized_version` della cartella `ch01`.

Riutilizzare questo progetto per gli esempi successivi

Potete seguire le stesse fasi descritte in questo paragrafo per creare un progetto diverso per ciascuna applicazione del libro. Tuttavia, per i nostri esempi vi consigliamo di riutilizzare questo progetto semplicemente rimuovendo il programma `GuessNumber.c` e aggiungendo poi un altro programma in C. Per rimuovere un file dal progetto (ma non dal vostro sistema), selezionatelo dalla finestra **Solution Explorer**, poi premete **Canc.** Ripetete quindi la *Fase 4* per aggiungere un nuovo programma al progetto.

Utilizzare Ubuntu Linux nel sottosistema Windows per Linux

Gli utenti Windows potrebbero scegliere di utilizzare il compilatore GNU gcc su Windows, in particolare per quei pochi programmi di questo libro che non possono essere compilati da Visual C++. È possibile farlo usando il contenitore Docker della GNU Compiler Collection (Paragrafo 1.10.4), oppure usando il compilatore gcc in Ubuntu Linux che viene eseguito nel sottosistema Windows per Linux. Per installare il sottosistema Windows per Linux, consultate le istruzioni all'indirizzo

<https://docs.microsoft.com/en-us/windows/wsl/install-win10>

Dopo aver installato e lanciato Ubuntu sul vostro sistema Windows, passate alla cartella contenente l'esempio di codice con il comando seguente:

`cd /mnt/c/Users/VostroNomeUtente/Documents/examples/ch01`

Proseguite poi con la *Fase 2* del Paragrafo 1.10.3.

1.10.2 Compilare ed eseguire un'applicazione in C con Xcode su macOS

In questo paragrafo eseguirete un programma in C su macOS usando il compilatore Clang nell'IDE Xcode di Apple.

Fase 1: controllare il setup

Se non l'avete già fatto, leggete nella sezione “Prima di cominciare” le istruzioni per installare l'IDE e scaricare gli esempi di codice del libro.

Fase 2: lanciare Xcode

Aprite una finestra del Finder, selezionate **Applications** e fate doppio clic sull'icona di Xcode (⌘). Se è la prima volta che eseguite Xcode, apparirà la finestra **Welcome to Xcode**; chiudetela facendo clic sulla **X** nell'angolo in alto a sinistra (potete riaprirla in qualunque momento con i comandi **Window > Welcome to Xcode**). Usiamo il simbolo **>** per indicare la selezione di una voce da un menu: per esempio, **File > Open** significa “selezionate la voce di menu **Open** dal menu **File**”.

Fase 3: creare un progetto

Un **progetto** consiste in un gruppo di file correlati, come i file di codice sorgente in C che compongono un'applicazione. I progetti Xcode presentati come esempi in questo libro sono di tipo **Command Line Tool** che eseguirete nell'IDE. Per creare un progetto:

1. Selezionate **File > New > Project...**
2. In cima alla finestra di dialogo **Choose a template for your new project**, fate clic su **macOS**.
3. Sotto **Application**, fate clic su **Command Line Tool** e quindi su **Next**.
4. Come **Product Name**, digitate un nome per il vostro progetto: nel nostro caso abbiamo indicato **C_test_Xcode**.
5. Nell'elenco a discesa **Language**, selezionate **C**, quindi fate clic su **Next**.
6. Specificate dove volete salvare il vostro progetto: noi abbiamo indicato la cartella **examples** che contiene gli esempi di codice del libro.
7. Fate clic su **Create**.

Xcode crea il vostro progetto e visualizza la **finestra dello spazio di lavoro**, che inizialmente mostra tre diverse aree: **area Navigator** (a sinistra), **area Editor** (al centro) e **area Utilities** (a destra).

Nell'area **Navigator**, in alto, ci sono le icone dei *navigatori* che si possono visualizzare lì. Per gli esempi di questo libro, lavorerete principalmente con:

- **Project** (□), che mostra tutti i file e le cartelle del vostro progetto;
- **Issue** (⚠), che mostra gli avvisi e i messaggi di errore generati dal compilatore.

Facendo clic sul pulsante di un navigatore viene visualizzato il pannello corrispondente.

L'area **Editor** al centro serve per gestire i progetti e per fare l'editing del codice sorgente; viene sempre mostrata nella vostra finestra dello spazio di lavoro. Selezionando un file nel navigatore **Project** viene visualizzato il contenuto del file nell'area **Editor**. Negli esempi del libro non è previsto l'uso dell'area **Utilities**. Utilizzerete l'area **Debug**, che apparirà sotto l'area **Editor**, per eseguire il programma “indovina il numero” e interagire con esso.

La barra degli strumenti della finestra dello spazio di lavoro contiene le opzioni per eseguire un programma, visualizzare lo stato di avanzamento delle attività in esecuzione su Xcode e mostrare o nascondere le aree che si trovano a sinistra (**Navigator**), a destra (**Utilities**) e in basso (**Debug**).

Fase 4: rimuovere il file `main.c` dal progetto

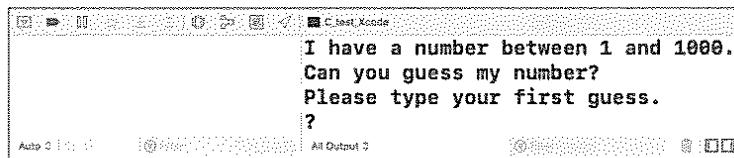
Xcode crea per default un file di codice sorgente `main.c` contenente un semplice programma che visualizza “Hello, World!”, ma in questo test guidato non userete `main.c`. Quindi, nel navigatore **Project**, fate clic con il tasto destro del mouse sul file `main.c` e selezionate **Delete**. Nella finestra di dialogo che appare, selezionate **Move to Trash** e svuotate il cestino per rimuovere il file dal vostro sistema.

Fase 5: aggiungere il file GuessNumber.c nel progetto

In una finestra del Finder, apriate la cartella ch01 contenuta nella cartella examples del libro, quindi trascinate GuessNumber.c nella cartella **C_Test_Xcode** del navigatore **Project**. Nella finestra di dialogo che appare, assicuratevi che sia selezionato **Copy items if needed** e poi fate clic su **Finish**.⁴⁰

Fase 6: compilare ed eseguire il progetto

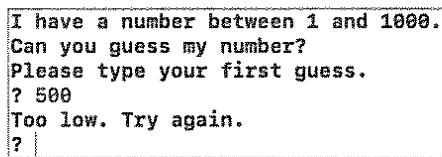
Per compilare ed eseguire il progetto in modo da poter testare l'applicazione, fate clic sul pulsante di esecuzione (▶) sulla barra degli strumenti di Xcode. Se il programma viene compilato correttamente, Xcode apre l'area **Debug** ed esegue il programma nella metà destra di tale area:



L'applicazione visualizza “Please type your first guess.”, poi visualizza un punto di domanda (?) come prompt sulla riga successiva.

Fase 7: inserire la prima risposta

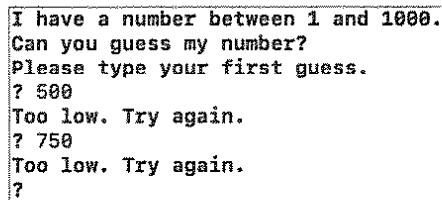
Fate clic nell'area **Debug**, poi digitate **500** e premete *Invio*:



L'applicazione visualizza “Too low. Try again.” per indicare che il valore inserito è minore del numero scelto dall'applicazione come risposta esatta.

Fase 8: inserire un'altra risposta

Al prompt successivo, digitate **750**:



L'applicazione visualizza “Too low. Try again.” per indicare che il valore inserito è ancora minore del numero scelto dall'applicazione come risposta esatta.

40. Nel caso di programmi con più file di codice sorgente che vedrete nei capitoli successivi, trascinate tutti i file per il programma considerato nella cartella del progetto. Quando comincerete a creare programmi per conto vostro, potrete fare clic con il tasto destro del mouse sulla cartella del progetto e selezionare **New File...** per visualizzare una finestra di dialogo che vi permetterà di aggiungere un nuovo file.

Fase 9: inserire ulteriori risposte

Continuate a giocare introducendo valori finché non indovinate il numero esatto. A quel punto l'applicazione mostrerà “Excellent! You guessed the number!”:

```
? 875
Too high. Try again.
? 812
Too high. Try again.
? 781
Too low. Try again.
? 797
Too low. Try again.
? 805
Too low. Try again.
? 808

Excellent! You guessed the number!
Would you like to play again?
Please type (1=yes, 2=no)?
```

Fase 10: continuare il gioco o uscire dall'applicazione

Dopo che è stato indovinato il numero corretto, l'applicazione domanda se volete continuare a giocare. Al prompt “Please type (1=yes, 2=no)?”, inserite **1** per giocare ancora, con un nuovo numero da indovinare, oppure inserite **2** per chiudere l'applicazione. Ogni volta che eseguirete questa applicazione (*Fase 6*), essa sceglierà gli stessi numeri da farvi indovinare. Per giocare con numeri a caso, usate la versione di `GuessNumber.c` che troverete nella sottocartella `randomized_version` nella cartella `ch01`.

Riutilizzare questo progetto per gli esempi successivi

Potete seguire le stesse fasi descritte in questo paragrafo per creare un progetto diverso per ciascuna applicazione del libro. Tuttavia, per gli altri esempi vi consigliamo di riutilizzare questo progetto semplicemente rimuovendo il programma corrente e aggiungendone uno nuovo. Per rimuovere un file dal progetto (ma non dal vostro sistema), fate clic con il tasto destro del mouse sul file nel navigatore **Project** e selezionate **Delete**, selezionando poi **Remove Reference** nella finestra di dialogo che apparirà. Ripetete quindi la *Fase 5* per aggiungere un nuovo programma al progetto.

1.10.3 Compilare ed eseguire un'applicazione in C con GNU gcc su Linux

Per questa modalità del test guidato, assumiamo che abbiate letto la sezione “Prima di cominciare” e che abbiate scaricato gli esempi di codice nella cartella `Documents` del vostro account utente.

Fase 1: passare alla cartella ch01

Da una shell Linux, usate il comando `cd` per passare alla sottocartella `ch01` della cartella `examples` del libro:

```
~$ cd ~/Documents/examples/ch01
~/Documents/examples/ch01$
```

Per le figure di questo paragrafo, useremo un carattere in **grassetto** per evidenziare il testo che dovrete digitare. Il prompt nella shell sul nostro sistema Ubuntu Linux usa il segno tipografico tilde (~) per rappresentare la home directory, e ogni prompt termina col carattere grafico dollaro (\$). Il prompt potrà variare nelle diverse distribuzioni Linux.

Fase 2: compilare l'applicazione

Prima di poter eseguire l'applicazione, dovete compilarla:

```
~/Documents/examples/ch01$ gcc -std=c18 GuessNumber.c -o GuessNumber
~/Documents/examples/ch01$
```

Il comando `gcc` compila l'applicazione:

- l'opzione `-std=c18` indica che stiamo utilizzando C18, la versione più recente dello standard del linguaggio di programmazione C;
- l'opzione `-o` assegna il nome al file eseguibile (`GuessNumber`) che userete per eseguire il programma.

Fase 3: eseguire l'applicazione

Digitate `./GuessNumber` al prompt e premete *Invio* per eseguire il programma:

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?
```

I caratteri `.` dicono a Linux di eseguire un file dalla directory corrente, e sono qui necessari per indicare che `GuessNumber` è un file eseguibile.

Fase 4: inserire la prima risposta

L'applicazione visualizza “`Please type your first guess.`” e poi un punto interrogativo (?) come prompt nella riga successiva. Al prompt, digitate **500** (a seconda del compilatore utilizzato, gli output potrebbero variare):

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
?
```

L'applicazione visualizza “`Too high. Try again.`” per indicare che il valore inserito è maggiore del numero scelto dall'applicazione come risposta esatta.

Fase 5: inserire un'altra risposta

Al prompt successivo, digitate **250**:

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too low. Try again.
?
```

Questa volta l'applicazione visualizza “`Too low. Try again.`”, poiché il valore inserito è minore di quello della risposta esatta.

Fase 9: inserire ulteriori risposte

Continuate a giocare introducendo valori finché non indovinate il numero esatto. A quel punto l'applicazione mostrerà "Excellent! You guessed the number!":

```
? 875
Too high. Try again.
? 812
Too high. Try again.
? 781
Too low. Try again.
? 797
Too low. Try again.
? 805
Too low. Try again.
? 808

Excellent! You guessed the number!
Would you like to play again?
Please type (1=yes, 2=no)?
```

Fase 10: continuare il gioco o uscire dall'applicazione

Dopo che è stato indovinato il numero corretto, l'applicazione domanda se volete continuare a giocare. Al prompt "Please type (1=yes, 2=no)?", inserite **1** per giocare ancora, con un nuovo numero da indovinare, oppure inserite **2** per chiudere l'applicazione. Ogni volta che eseguirete questa applicazione (*Fase 6*), essa sceglierà gli stessi numeri da farvi indovinare. Per giocare con numeri a caso, usate la versione di `GuessNumber.c` che troverete nella sottocartella `randomized_version` nella cartella `ch01`.

Riutilizzare questo progetto per gli esempi successivi

Potete seguire le stesse fasi descritte in questo paragrafo per creare un progetto diverso per ciascuna applicazione del libro. Tuttavia, per gli altri esempi vi consigliamo di riutilizzare questo progetto semplicemente rimuovendo il programma corrente e aggiungendone uno nuovo. Per rimuovere un file dal progetto (ma non dal vostro sistema), fate clic con il tasto destro del mouse sul file nel navigatore **Project** e selezionate **Delete**, selezionando poi **Remove Reference** nella finestra di dialogo che apparirà. Ripetete quindi la *Fase 5* per aggiungere un nuovo programma al progetto.

1.10.3 Compilare ed eseguire un'applicazione in C con GNU gcc su Linux

Per questa modalità del test guidato, assumiamo che abbiate letto la sezione "Prima di cominciare" e che abbiate scaricato gli esempi di codice nella cartella `Documents` del vostro account utente.

Fase 1: passare alla cartella ch01

Da una shell Linux, usate il comando `cd` per passare alla sottocartella `ch01` della cartella `examples` del libro:

```
~$ cd ~/Documents/examples/ch01
~/Documents/examples/ch01$
```

Per le figure di questo paragrafo, useremo un carattere in grassetto per evidenziare il testo che dovete digitare. Il prompt nella shell sul nostro sistema Ubuntu Linux usa il segno tipografico tilde (~) per rappresentare la home directory, e ogni prompt termina col carattere grafico dollaro (\$). Il prompt potrà variare nelle diverse distribuzioni Linux.

Fase 2: compilare l'applicazione

Prima di poter eseguire l'applicazione, dovete compilarla:

```
~/Documents/examples/ch01$ gcc -std=c18 GuessNumber.c -o GuessNumber
~/Documents/examples/ch01$
```

Il comando `gcc` compila l'applicazione:

- L'opzione `-std=c18` indica che stiamo utilizzando C18, la versione più recente dello standard del linguaggio di programmazione C;
- L'opzione `-o` assegna il nome al file eseguibile (`GuessNumber`) che userete per eseguire il programma.

Fase 3: eseguire l'applicazione

Digitate `./GuessNumber` al prompt e premete *Invio* per eseguire il programma:

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?
```

I caratteri `.` dicono a Linux di eseguire un file dalla directory corrente, e sono qui necessari per indicare che `GuessNumber` è un file eseguibile.

Fase 4: inserire la prima risposta

L'applicazione visualizza "Please type your first guess." e poi un punto interrogativo (?) come prompt nella riga successiva. Al prompt, digitate **500** (a seconda del compilatore utilizzato, gli output potrebbero variare):

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
?
```

L'applicazione visualizza "Too high. Try again." per indicare che il valore inserito è maggiore del numero scelto dall'applicazione come risposta esatta.

Fase 5: inserire un'altra risposta

Al prompt successivo, digitate **250**:

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too low. Try again.
?
```

Questa volta l'applicazione visualizza "Too low. Try again.", poiché il valore inserito è minore di quello della risposta esatta.

Fase 6: inserire ulteriori risposte

Continuate a giocare introducendo valori finché non indovinate il numero esatto. A quel punto l'applicazione mostrerà “Excellent! You guessed the number!”:

```
Too low. Try again.
? 375
Too low. Try again.
? 437
Too high. Try again.
? 406
Too high. Try again.
? 391
Too high. Try again.
? 383
Too low. Try again.
? 387
Too high. Try again.
? 385
Too high. Try again.
? 384

Excellent! You guessed the number!
Would you like to play again?
Please type (1=yes, 2=no)?
```

Fase 7: continuare il gioco o uscire dall'applicazione

Dopo che è stato indovinato il numero corretto, l'applicazione domanda se volete continuare a giocare. Al prompt “Please type (1=yes, 2=no)?”, inserite **1** per giocare ancora, con un nuovo numero da indovinare, oppure inserite **2** per chiudere l'applicazione. Ogni volta che eseguirete questa applicazione (*Fase 3*), essa sceglierà gli stessi numeri da farvi indovinare. Per giocare con numeri a caso, usate la versione di `GuessNumber.c` che troverete nella sottocartella `randomized_version` nella cartella `ch01`.

1.10.4 Compilare ed eseguire un'applicazione in C usando un contenitore Docker della GNU Compiler Collection (GCC) in esecuzione nativa su Windows 10, macOS o Linux

Una delle migliori modalità multiplattforma per eseguire il compilatore GNU `gcc` è tramite il contenitore Docker della GNU Compiler Collection (GCC). Per questo paragrafo assumiamo che abbiate installato Docker Desktop (su Windows o macOS) oppure Docker Engine (su Linux), come spiegato nella sezione “Prima di cominciare”, e che Docker sia in esecuzione sul vostro computer.

Eseguire il contenitore Docker della GNU Compiler Collection (GCC)

Aprite un prompt dei comandi (Windows), una finestra di Terminale (macOS/Linux) o una shell (Linux), poi procedete come segue per lanciare il contenitore Docker della GCC:

1. Usate il comando `cd` per andare fino alla cartella `examples` contenente gli esempi di questo libro; lanciando da qui il contenitore Docker, gli consentirete di accedere ai nostri esempi di codice.
2. **Utenti Windows:** lanciate il contenitore Docker della GCC con il comando⁴¹

```
docker run --rm -it -v "%CD%":/usr/src gcc:latest
```

41. Comparirà un messaggio di notifica con la richiesta di permettere a Docker l'accesso ai file nella cartella corrente: dovete consentire, altrimenti non potrete accedere ai nostri file di codice sorgente in Docker.

3. Utenti macOS/Linux:

lanciate il contenitore Docker della GCC con il comando

```
docker run --rm -it -v "$(pwd)":/usr/src gcc:latest
```

Nei comandi precedenti:

- `--rm` “pulisce” le risorse del contenitore della GCC quando alla fine lo chiuderete;
- `-it` esegue il contenitore in modalità interattiva, e vi permette quindi di inserire i comandi per cambiare cartella, di compilare i programmi usando il compilatore GNU `gcc` e di eseguirli;
- il comando `-v "%CD%":/usr/src` (Windows) oppure `-v "$(pwd)":/usr/src` (macOS/Linux) consente al contenitore Docker di accedere ai file e alle sottocartelle contenuti nella vostra cartella corrente tramite la cartella `/usr/src` del contenitore Docker. Con il comando `cd` potete andare alle sottocartelle di `/usr/src` per compilare ed eseguire i nostri programmi;
- `gcc:latest` è il nome del contenitore da voi installato nella sezione “Prima di cominciare”.⁴²

Quando il contenitore sarà in esecuzione, vedrete un prompt simile al seguente

```
root@67773f59d9ea:/#
```

ma sul vostro computer la parte “@67773f59d9ea” sarà differente. Il sistema operativo Linux del contenitore utilizza il carattere slash (/) come separatore tra le cartelle; il prompt visualizza la posizione della cartella corrente tra i caratteri : e #.

Passare alla cartella ch01 nel contenitore Docker

Usate il comando `cd` per passare alla cartella `/usr/src/ch01`:

```
root@01b4d47cad6:/# cd /usr/src/ch01
root@01b4d47cad6:/usr/src/ch01#
```

Potete ora compilare, eseguire e interagire con l’applicazione `GuessNumber` nel contenitore Docker, utilizzando i comandi visti nel Paragrafo 1.10.3, *Fasi 2-7*.

Terminare il contenitore Docker

Per terminare il contenitore Docker digitate `Ctrl + d` al prompt del contenitore stesso.

1.11 Internet, World Wide Web, cloud e IoT

Alla fine degli anni Sessanta, la Advanced Research Projects Agency (ARPA) del Dipartimento della Difesa degli Stati Uniti mise a punto un progetto per collegare in rete i principali sistemi informatici di circa una dozzina di università e istituti di ricerca da essa finanziati. I computer dovevano essere collegati con linee di comunicazione che operavano a una velocità dell’ordine di 50.000 bit al secondo, una velocità stupefacente in un momento in cui la maggior parte delle persone (le poche che avevano accesso alla rete) si connetteva ai computer tramite linee telefoniche alla velocità di 110 bit al secondo. La ricerca accademica era sul punto di fare un gigantesco passo avanti. ARPA procedette a implementare ciò che rapidamente divenne noto come ARPANET, il precursore di quello che oggi è Internet. Attualmente le più elevate velocità di Internet sono nell’ordine di miliardi di bit al secondo, con una prospettiva imminente di un trilione di bit al secondo!⁴³ Nel 2020, i ricercatori australiani hanno testato con successo una connessione Internet da 44,2 terabit al secondo.⁴⁴

42. `gcc:latest` è il nome della versione più recente del contenitore Docker della GCC nel momento in cui lo avete scaricato nel vostro sistema. Una volta installato, il contenitore non si aggiorna automaticamente. Per mantenerlo aggiornato con l’ultima versione disponibile dovete eseguire `docker pull gcc:latest`; nel caso ci fosse una nuova versione, Docker la scaricherà.

43. “BT Testing 1.4 Terabit Internet Connections.” Accesso 1 novembre 2020. <https://testinternetspeed.org/blog/bt-testing-1-4-terabit-internet-connections/>.

44. “Monash, Swinburne, and RMIT universities use optical chip to achieve 44Tbps data speed.” Accesso 9 gennaio 2021. <https://www.zdnet.com/article/monash-swinburne-and-rmit-universities-achieve-44tbps-data-speed-using-single-optical-chip/>.

Le cose sono andate diversamente dal progetto originario. Sebbene ARPANET abbia permesso ai ricercatori di collegare in rete i loro computer, la sua principale utilità è stata quella di consentire una forma di comunicazione semplice e rapida attraverso quella che oggi è chiamata posta elettronica (e-mail). Questo vale anche per l'attuale Internet, in quanto le e-mail, l'instant messaging, il trasferimento di file e i social media come Snapchat, Instagram, Facebook e Twitter permettono a miliardi di persone in tutto il mondo di comunicare rapidamente e facilmente.

Il protocollo (un insieme di regole) per comunicare su ARPANET divenne noto come **TCP (Transmission Control Protocol)**. Il TCP assicurava che i messaggi, costituiti da parti numerate in sequenza chiamate pacchetti, venissero instradati in modo corretto dal mittente al destinatario, arrivassero intatti e fossero assemblati nell'ordine corretto.

1.11.1 Internet: una rete di reti

In parallelo con l'iniziale evoluzione di Internet, varie organizzazioni in tutto il mondo implementavano le proprie reti sia per la comunicazione intraorganizzativa (ossia all'interno di un'organizzazione) che per quella interorganizzativa (ossia tra diverse organizzazioni). Apparve una grande varietà di hardware e software di rete. Una sfida era permettere a queste diverse reti di comunicare tra loro. ARPA raggiunse l'obiettivo sviluppando l'**IP (Internet Protocol)**, che creava effettivamente una rete di network, l'attuale architettura di Internet. L'insieme combinato di protocolli è ora chiamato **TCP/IP**. Ogni dispositivo connesso a Internet ha un **indirizzo IP**, un identificatore numerico univoco che permette ai dispositivi che comunicano via TCP/IP di individuarsi l'un l'altro in Internet.

Le aziende si resero conto rapidamente che utilizzando Internet avrebbero potuto migliorare la proprie attività e offrire servizi nuovi e migliori ai propri clienti. Perciò iniziarono a investire grandi somme di denaro per sviluppare e migliorare la propria presenza su Internet. Questo ha generato una forte concorrenza tra le compagnie di telecomunicazione come tra i produttori di hardware e software per soddisfare la crescente domanda di infrastrutture. Di conseguenza, la larghezza di banda (la capacità delle linee di comunicazione di trasportare informazioni) su Internet è aumentata enormemente, mentre i costi dell'hardware sono crollati.

1.11.2 World Wide Web: come rendere Internet user-friendly

Il **World Wide Web** (chiamato semplicemente "Web") è una raccolta di hardware e software associati a Internet che consente agli utenti di computer di trovare e visualizzare documenti (con varie combinazioni di testo, grafica, animazioni, audio e video) su quasi ogni argomento. Nel 1989, Tim Berners-Lee del CERN (Centro Europeo per la Ricerca Nucleare) iniziò a sviluppare il linguaggio **HTML (HyperText Markup Language)**, la tecnologia per condividere informazioni attraverso documenti dotati di "collegamenti ipertestuali", e a progettare protocolli di comunicazione come l'**HTTP (HyperText Transfer Protocol)** che hanno costituito le basi del suo nuovo sistema informativo, a cui attribui il nome di World Wide Web.

Nel 1994, Berners-Lee fondò il **W3C (World Wide Web Consortium, <https://www.w3.org>)**, dedicato allo sviluppo di tecnologie web. Uno degli obiettivi principali di W3C è di rendere il Web accessibile a tutti, senza limitazioni dovute a disabilità, linguaggio o cultura.

1.11.3 Cloud

Oggi giorno l'informatica è fatta sempre di più "nel cloud", ovvero usando software e dati distribuiti da una parte all'altra del mondo attraverso Internet anziché localizzati sul proprio computer desktop, notebook o dispositivo mobile. Il **cloud computing** consente di aumentare o diminuire le risorse di calcolo per soddisfare le proprie esigenze in ogni momento: una soluzione più conveniente rispetto all'acquisto di hardware per fornire spazio di memoria e potenza di elaborazione sufficienti per soddisfare gli occasionali picchi di richiesta. Il cloud computing inoltre consente di risparmiare denaro spostando sul provider di servizi l'onere di gestione delle applicazioni (come installazione e aggiornamento del software, sicurezza, backup e *disaster recovery*).

Le applicazioni che usiamo ogni giorno sono largamente dipendenti da diversi **servizi basati su cloud**, che si avvalgono di enormi agglomerati di risorse computazionali (computer, processori, memorie, dischi ecc.) e database che comunicano tra loro e con le app attraverso Internet. Un servizio a cui si può accedere tramite Internet è detto **servizio web**.

Software “as a Service”

I fornitori di servizi cloud si orientano principalmente sulla tecnologia SOA (*Service-Oriented Architecture*), un’architettura orientata ai servizi. Forniscono funzionalità “as a Service” a cui le applicazioni possono connettersi e che possono usare nel cloud. Tra i servizi comunemente offerti dai fornitori di cloud troviamo:⁴⁵

Acronimi “as-a-Service” (alcuni sono uguali)	
Big Data as a Service (BDaaS)	Platform as a Service (PaaS)
Hadoop as a Service (HaaS)	Software as a Service (SaaS)
Infrastructure as a Service (IaaS)	Storage as a Service (SaaS)

Mashup

La metodologia di sviluppo delle applicazioni **mashup** consente di sviluppare rapidamente applicazioni software potenti combinando servizi web complementari (spesso gratuiti) con altre forme di feed di informazioni. Uno dei primi mashup, www.housingmaps.com, combinava gli elenchi di appartamenti di Craigslist (www.craigslist.org) con le funzionalità di mappatura di Google Maps per visualizzare la posizione delle case in vendita o in affitto in una determinata area. Date un’occhiata al sito www.housingmaps.com per alcuni fatti interessanti, storia, articoli e modalità con cui ha influenzato gli elenchi del settore immobiliare.

In ProgrammableWeb (<https://programmableweb.com/>) potrete trovare una directory di circa 24.000 servizi web e quasi 8.000 mashup. Vi sono anche istruzioni ed esempi di codice per lavorare con servizi web e creare i vostri mashup. Secondo il loro sito web, tra i servizi web più utilizzati vi sono Google Maps e altri forniti da Facebook, Twitter e YouTube.

1.11.4 Internet delle cose

Internet non è più soltanto una rete di *computer*, è diventato un **Internet delle cose** (IoT, *Internet of Things*). Si considera *cosa* qualunque oggetto con un indirizzo IP e la capacità di inviare, e in alcuni casi ricevere, automaticamente dati attraverso Internet. Queste *cose* includono:

- automobili con un dispositivo per pagare i pedaggi;
- monitor per la disponibilità di posti auto in un garage;
- monitor cardiaci impiantati in esseri umani;
- rilevatori della qualità dell’acqua;
- contatori smart che misurano l’energia utilizzata;
- rilevatori di radiazioni;
- localizzatori di articoli nei magazzini;
- app mobili che possono tracciare i vostri spostamenti e la vostra posizione;
- termostati smart che regolano la temperatura della stanza in base alle previsioni del tempo e alle attività nella casa;
- elettrodomestici intelligenti.

Secondo statista.com, ci sono già più di 23 miliardi di dispositivi IoT in uso e si prevede che ce ne saranno più di 75 miliardi nel 2025.⁴⁶

✓ Autovalutazione

1. (Completare) Il precursore di quello che oggi è Internet è stato _____.

Risposta: ARPANET.

45. Per ulteriori acronimi “as-a-Service”, consultate https://en.wikipedia.org/wiki/Cloud_computing e https://en.wikipedia.org/wiki/As_a_service.

46. “Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025.” Accesso 1 novembre 2020. <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>.

2. (*Completare*) Il _____ (detto semplicemente “il Web”) è una raccolta di hardware e software associati a Internet che consente agli utenti di computer di trovare e visualizzare documenti (con varie combinazioni di testo, grafica, animazioni, audio e video).

Risposta: World Wide Web.

3. (*Completare*) Nell’Internet delle cose (IoT), si considera *cosa* qualunque oggetto con un _____ e la capacità di inviare, e in alcuni casi ricevere, automaticamente dati attraverso Internet.

Risposta: indirizzo IP.

1.12 Tecnologie software

Se studiate e lavorate nel campo dello sviluppo software incontrerete spesso i termini elencati di seguito.

- **Refactoring:** tecnica che riguarda la rielaborazione di programmi al fine di renderli più chiari e più facili da gestire, preservando al tempo stesso la loro correttezza e funzionalità. Molti IDE contengono *strumenti di refactoring* in grado di eseguire automaticamente buona parte del processo di rielaborazione.
- **Design pattern:** architetture già sperimentate per la costruzione di software orientato agli oggetti, che sia flessibile e facile da gestire. Il settore dei design pattern cerca di catalogare alcuni modelli ricorrenti e incoraggia i progettisti software a *riutilizzarli* per sviluppare software di migliore qualità, con l’impiego di minor tempo, denaro e fatica.
- **Software Development Kits (SDK):** gli strumenti e la documentazione usati dagli sviluppatori per realizzare applicazioni.

✓ Autovalutazione

1. (*Completare*) Il _____ è la tecnica che riguarda la rielaborazione di programmi al fine di renderli più chiari e più facili da gestire, preservando al tempo stesso la loro correttezza e funzionalità.

Risposta: refactoring.

1.13 Quanto sono grandi i big data?

Per l’informatica e per la data science, ora i dati sono importanti tanto quanto la scrittura dei programmi. Secondo IBM, circa 2,5 trilioni di byte (2,5 *exabyte*) di dati vengono creati ogni giorno⁴⁷ e il 90% dei dati mondiali è stato creato negli ultimi due anni.⁴⁸ Internet, che giocherà un ruolo importante nella vostra carriera, è responsabile di gran parte di questa tendenza. Secondo IDC, la fornitura mondiale di dati raggiungerà 175 *zettabyte* (equivalenti a 175 migliaia di miliardi di gigabyte o 175 miliardi di terabyte) annuali entro il 2025.⁴⁹ Di seguito esamineremo alcune delle misure di dati più popolari.

Megabyte (MB)

Un megabyte è all’incirca un milione di byte (2²⁰ byte per essere precisi). Molti dei file che si usano comunemente richiedono uno o più MB di memoria. Ecco alcuni esempi.

- **File audio MP3:** gli MP3 di alta qualità occupano da 1 a 2,4 MB al minuto.⁵⁰
- **Fotografie:** il formato JPEG per fotografie realizzate con fotocamere digitali richiede all’incirca da 8 a 10 MB per foto.

47. “Welcome to the world of A.I.” Accesso 1 novembre 2020. <https://www.ibm.com/blogs/watson/2016/06/welcome-to-the-world-of-a-i/>.

48. “Accelerate Research and Discovery.” Accesso 1 novembre 2020. <https://www.ibm.com/watson/advantages/accelerate>.

49. “IDC: Expect 175 zettabytes of data worldwide by 2025.” Accesso 1 novembre 2020. <https://www.networkworld.com/article/3325397/storage/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html>.

50. “Audio File Size Calculations.” Accesso 1 novembre 2020. <https://www.audiomountain.com/tech/audio-file-size.html>.

- Video: le videocamere dei cellulari possono registrare video a diverse risoluzioni. Ogni minuto di video può richiedere molti megabyte di memoria. Per esempio, nelle impostazioni dell'app **Camera** di un iPhone è indicato che un video 1080p a 30 fotogrammi al secondo (FPS) necessita di 130 MB al minuto, mentre un video 4K a 30 FPS necessita di 350 MB al minuto.

Gigabyte (GB)

Un gigabyte equivale circa a 1.000 megabyte (2^{30} per essere precisi). Un DVD dual-layer può memorizzare fino a 8,5 GB⁵¹, la stessa quantità di:

- fino a 141 ore di file audio MP3;
- approssimativamente 1.000 foto di una fotocamera da 16 megapixel;
- approssimativamente 7,7 minuti di un video 1080p a 30 FPS;
- approssimativamente 2,85 minuti di un video 4K a 30 FPS.

I dischi Blu-ray Ultra HD possono memorizzare fino a 100 GB di video.⁵² La visione di filmati 4K in streaming può richiedere dai 7 ai 10 GB all'ora (con compressione alta).

Terabyte (TB)

Un terabyte è uguale a circa 1.000 gigabyte (2^{40} byte per essere precisi). Gli attuali dischi per personal computer arrivano fino a 20 TB,⁵³ che è equivalente a:

- approssimativamente 28 anni di file audio MP3;
- approssimativamente 1,68 milioni di fotografie di una fotocamera da 16 megapixel;
- approssimativamente 226 ore di un video 1080p a 30 FPS;
- approssimativamente 84 ore di un video 4K a 30 FPS.

Il più grande disco a stato solido (SSD) di Nimbus Data ha una capacità di 100 TB e può quindi memorizzare 5 volte la quantità di audio, foto e video degli esempi precedenti (20 TB).⁵⁴

Petabyte, exabyte e zettabyte

Online ci sono circa 4 miliardi di persone che generano più o meno 2,5 trilioni di dati ogni giorno,⁵⁵ ovvero 2.500 petabyte (ogni petabyte è circa 1.000 terabyte) o 2,5 exabyte (ogni exabyte è circa 1.000 petabyte). Secondo un articolo di *Analytics-Week* del marzo 2016, entro il 2021 ci sarebbero stati più di 50 miliardi di dispositivi connessi a Internet (molti dei quali attraverso l'Internet delle cose, esaminato nel Paragrafo 1.11.4) ed entro il 2020 sarebbero stati prodotti 1,7 megabyte di nuovi dati ogni secondo *per ogni persona sul pianeta*.⁵⁶ In questo momento (con approssimativamente 7,7 miliardi di persone⁵⁷) sono circa:

- 13 petabyte di nuovi dati al secondo;
- 780 petabyte al minuto;
- 46.800 petabyte (46,8 exabyte) all'ora;
- 1.123 exabyte al giorno, ovvero 1,123 zettabyte (ZB) al giorno (ogni zettabyte corrisponde a circa 1.000 exabyte).

È l'equivalente di circa 5,5 milioni di ore (più di 600 anni) di video 4K al giorno, oppure di 116 miliardi di foto al giorno!

51. "DVD." Accesso 1 novembre 2020. <https://en.wikipedia.org/wiki/DVD>.

52. "Ultra HD Blu-ray." Accesso 1 novembre 2020. https://en.wikipedia.org/wiki/Ultra_HD_Blu-ray.

53. "History of hard disk drives." Accesso 1 novembre 2020. https://en.wikipedia.org/wiki/History_of_hard_disk_drives.

54. "Nimbus Data 100TB SSD – World's Largest SSD." Accesso 1 novembre 2020. <https://www.cinema5d.com/nimbus-data-100tb-ssd-worlds-largest-ssd/>.

55. "How Much Data Is Created Every Day in 2020?" Accesso 1 novembre 2020. <https://techjury.net/blog/how-much-data-is-created-every-day/#gref>.

56. "Big Data Facts." Accesso 1 novembre 2020. <https://analyticsweek.com/content/big-data-facts/>.

57. "World Population." Accesso 1 novembre 2020. https://en.wikipedia.org/wiki/World_population.

Altre statistiche sui big data

Per avere un'idea in tempo reale delle quantità relative ai big data, visitate <https://www.internetlivestats.com>, che riporta diverse statistiche, tra le quali:

- il numero di ricerche su Google;
- il numero di tweet;
- il numero di video visti su YouTube;
- il numero di foto caricate su Instagram.

Nel sito potrete accedere a informazioni aggiuntive per ogni statistica.

Di seguito sono elencati altri fatti interessanti relativi ai big data:

- ogni ora, gli utenti YouTube caricano 30.000 ore di video, mentre le ore di video viste su YouTube ogni giorno sono circa 1 miliardo;⁵⁸
- ogni secondo, su Internet c'è un traffico di 103.777 GB (o 103,777 TB), vengono inviati 9.204 tweet, effettuate 87.015 ricerche su Google e visti 86.617 video su YouTube;⁵⁹
- ogni giorno, su Facebook ci sono 3,2 miliardi di "like" e commenti,⁶⁰ e vengono inviati 5 miliardi di emoji con Facebook Messenger.⁶¹

Un'infografica interessante, chiamata "Data Never Sleeps 8.0", è offerta su Domo Inc. e mostra quanti dati sono generati *ogni minuto*:⁶²

- 347.222 post su Instagram;
- 500 ore di video caricate su YouTube;
- 147.000 foto caricate su Facebook;
- 41.666.667 messaggi WhatsApp condivisi;
- 404.444 ore di video viste su Netflix;
- 479.452 utenti interagiscono con i contenuti di Reddit;
- 208.333 utenti partecipano alle riunioni Zoom;
- 1.388.889 utenti effettuano videochiamate.

Fonte: <https://www.domo.com/learn/data-never-sleeps-8>

Risorse computazionali

Così come i dati, anche la potenza dei computer per elaborarli sta aumentando. Le prestazioni dei moderni processori vengono misurate in termini di **FLOPS** (*floating-point operations per second, operazioni in virgola mobile al secondo*). Nella prima metà degli anni Novanta del secolo scorso, le capacità dei supercomputer più veloci venivano misurate in gigaflop (10^9 FLOP). A fine anni Novanta, Intel produsse il primo supercomputer a teraflop (10^{12} FLOP). Nella prima metà degli anni Duemila, la velocità raggiunse le centinaia di teraflop. Poi, nel 2008, IBM produsse il primo supercomputer a petaflop (10^{15} FLOP). Attualmente il supercomputer più veloce, Fugaku di Fujitsu⁶³, ha una capacità di 442 petaflop.⁶⁴

Utilizzando la computazione distribuita si possono collegare migliaia di computer attraverso Internet per produrre ancora più FLOP. A fine 2016, la rete Folding@home – una rete distribuita nella quale le persone mettono a disposizione le risorse dei propri computer perché vengano utilizzate nella ricerca su malattie o nella

58. "57 Fascinating and Incredible YouTube Statistics." Accesso 1 novembre 2020. <https://www.brandwatch.com/blog/youtube-stats/>.

59. "Tweets Sent in 1 Second." Accesso 1 novembre 2020. <http://www.internetlivestats.com/one-second>.

60. "Facebook: 3.2 Billion Likes & Comments Every Day." Accesso 1 novembre 2020. <https://marketingland.com/facebook-3-2-billion-likes-comments-every-day-19978>.

61. "Facebook celebrates World Emoji Day by releasing some pretty impressive facts." Accesso 1 novembre 2020. <https://mashable.com/2017/07/17/facebook-world-emoji-day/>.

62. "Data Never Sleeps 8.0." Accesso 1 novembre 2020. <https://www.domo.com/learn/data-never-sleeps-8>.

63. "Top 500." Accesso 24 dicembre 2020. https://en.wikipedia.org/wiki/TOP500#TOP_500.

64. "FLOPS." Accesso 1 novembre 2020. <https://en.wikipedia.org/wiki/FLOPS>.

progettazione di farmaci – raggiunse più di 100 petaflop.⁶⁵ Aziende come IBM stanno ora lavorando a supercomputer con capacità nell'ordine degli exaflop (10^{18} FLOP).⁶⁶

I computer quantistici ora in fase di sviluppo teoricamente potrebbero operare a una velocità 18.000.000.000.000.000 di volte superiore a quella dei moderni computer “convenzionali”!⁶⁷ Questo numero così sbalorditivo significa che un computer quantistico, teoricamente, in un secondo potrebbe fare più calcoli di quanti ne siano stati fatti dalla comparsa del primo computer a oggi. Questa potenza quasi inimmaginabile potrebbe mettere in serio pericolo le criptovalute basate su blockchain come i Bitcoin. Gli ingegneri stanno già ripensando l'architettura della blockchain⁶⁸ per prepararsi a questo enorme aumento di potenza computazionale.⁶⁹

La storia dei supercomputer ci dimostra che lo straordinario ammontare di denaro speso nei laboratori di ricerca per raggiungere queste prestazioni si concretizza poi in sistemi commerciali dal “prezzo ragionevole” o anche in personal computer, tablet e smartphone.

Il costo della potenza computazionale continua a decrescere, specialmente grazie al cloud. Un tempo ci si domandava: “Quanta potenza deve avere il mio sistema per reggere il *massimo* carico di elaborazione di cui ho bisogno?”. Oggi, questo modo di pensare è cambiato in: “Posso ottenere velocemente dal cloud le risorse *temporanee* che mi servono per risolvere questo compito gravoso?”. Si paga solamente quello che si utilizza per raggiungere un obiettivo.

Elaborare i dati del mondo richiede molta elettricità

Per elaborare l'esplosione di dati generati da tutti i dispositivi del mondo connessi a Internet serve un'enorme quantità di energia. Secondo un recente articolo, l'energia usata per l'elaborazione di dati sta crescendo del 20% all'anno e consuma dal tre al cinque per cento dell'energia mondiale. L'articolo stima che per il 2025 il totale dell'energia consumata per elaborare dati raggiungerà il 20%.⁷⁰

Una tecnologia che consuma enormi quantità di energia è la criptovaluta basata su blockchain Bitcoin. Generare una sola transazione di Bitcoin utilizza approssimativamente la stessa quantità di energia utilizzata in una settimana da una comune casa americana. Questo consumo è dovuto al processo utilizzato per validare la transazione.⁷¹

Secondo alcune stime, un anno di transazioni di Bitcoin consuma più energia di alcuni Paesi.⁷² Insieme, Bitcoin ed Ethereum (un'altra criptovaluta basata su blockchain) consumano annualmente più energia di Finlandia, Belgio o Pakistan.⁷³

Nel 2018 la Morgan Stanley disse che “il consumo di elettricità richiesto quest'anno per la creazione di criptovalute potrebbe effettivamente superare la richiesta mondiale di veicoli elettrici stimata dall'azienda nel 2025”.⁷⁴ Questa situazione non è sostenibile, specialmente perché c'è un enorme interesse nelle applicazioni basate su blockchain a prescindere dalle criptovalute. La comunità interessata alle blockchain sta lavorando per trovare soluzioni.^{75,76}

65. “Folding@home.” Accesso 1 novembre 2020. <https://en.wikipedia.org/wiki/Folding@home>.

66. “A new supercomputing-powered weather model may ready us for Exascale.” Accesso 1 novembre 2020. <https://www.ibm.com/blogs/research/2017/06/supercomputing-weather-model-exascale/>.

67. “Only God can count that fast — the world of quantum computing.” Accesso 1 novembre 2020. <https://medium.com/@n.biedrzycki/only-god-can-count-that-fast-the-world-of-quantum-computing-406a0a91fcf4>.

68. “Blockchain.” Accesso 24 dicembre 2020. <https://en.wikipedia.org/wiki/Blockchain>.

69. “Is Quantum Computing an Existential Threat to Blockchain Technology?” Accesso 1 novembre 2020. <https://singularityhub.com/2017/11/05/is-quantum-computing-an-existential-threat-to-blockchain-technology/>.

70. ““Tsunami of data” could consume one fifth of global electricity by 2025.” Accesso 1 novembre 2020. <https://www.theguardian.com/environment/2017/dec/11/tsunami-of-data-could-consume-fifth-global-electricity-by-2025>.

71. “One Bitcoin Transaction Consumes As Much Energy As Your House Uses in a Week.” Accesso 1 novembre 2020. https://motherboard.vice.com/en_us/article/ywbbpm/bitcoin-mining-electricity-consumption-ethereum-energy-climate-change.

72. “Bitcoin Energy Consumption Index.” Accesso 1 novembre 2020. <https://digiconomist.net/bitcoin-energy-consumption>.

73. “Ethereum Energy Consumption Index.” Accesso 1 novembre 2020. <https://digiconomist.net/ethereum-energy-consumption>.

74. “Power Play: What Impact Will Cryptocurrencies Have on Global Utilities?” Accesso 1 novembre 2020. <https://www.morganstanley.com/ideas/cryptocurrencies-global-utilities>.

75. “Blockchains Use Massive Amounts of Energy—But There's a Plan to Fix That.” Accesso 1 novembre 2020. <https://www.technologyreview.com/s/609480/bitcoin-uses-massive-amounts-of-energybut-theres-a-plan-to-fix-it/>.

76. “How to fix Bitcoin's energy-consumption problem.” Accesso 1 novembre 2020. <http://mashable.com/2017/12/01/bitcoin-energy/>.

Opportunità offerte dai big data

È probabile che l'esplosione dei big data continui a crescere esponenzialmente per diversi anni. Con 50 miliardi di dispositivi all'orizzonte, possiamo solo immaginare quanti altri ne arriveranno nei prossimi decenni.

È fondamentale per le aziende, per i governi, per l'esercito e anche per i singoli individui poter assimilare tutti questi dati. È interessante notare che alcune delle migliori ricerche su big data, data science e intelligenza artificiale provengono da organizzazioni economiche come J.P. Morgan, McKinsey, Bloomberg e altre. Grazie ai traguardi ottenuti rapidamente, è innegabile l'interesse per i big data da parte dei colossi finanziari. Molte aziende stanno facendo grossi investimenti per ottenere preziosi risultati usando tecnologie come big data, machine learning e l'elaborazione del linguaggio naturale. Questo forza anche le aziende concorrenti a investire, incrementando rapidamente il bisogno di professionisti con esperienza in informatica e data science. È una crescita destinata a continuare per diversi anni.

✓ Autovalutazione

1. (*Completare*) Le prestazioni dei processori sono misurate in termini di _____.

Risposta: FLOP (operazioni in virgola mobile al secondo).

2. (*Completare*) La tecnologia che potrebbe mettere in serio pericolo le criptovalute, come i Bitcoin, e altre tecnologie basate sulle blockchain è il _____.

Risposta: computer quantistico.

3. (*Vero/Falso*) Nel cloud si paga un prezzo fisso per i servizi a prescindere da quanto vengano effettivamente utilizzati?

Risposta: Falso. Uno dei vantaggi basilari del cloud è pagare solo per le risorse utilizzate.

1.13.1 Analisi dei big data

L'analisi dei dati è una disciplina accademica e professionale matura e ben sviluppata. Il termine "analisi dei dati" fu coniato nel 1962,⁷⁷ sebbene usare la statistica per analizzare i dati sia un'attività millenaria che risale agli antichi Egizi.⁷⁸ L'analisi dei big data è un fenomeno più recente, e il termine stesso "big data" è stato coniato intorno al 1987.⁷⁹

Elenchiamo le quattro V dei big data^{80,81}:

1. Volume: la quantità di dati prodotti nel mondo sta crescendo esponenzialmente.
2. Velocità: la velocità alla quale i dati vengono prodotti, la velocità alla quale si muovono e la velocità alla quale cambiano stanno tutte aumentando rapidamente.^{82,83,84}
3. Varietà: nel passato i dati erano alfanumerici (ovvero consistevano di caratteri alfabetici, cifre, punteggiatura e qualche carattere speciale), ora includono immagini, audio, video e dati provenienti da un numero crescente di sensori di Internet delle cose presenti in case private, aziende, veicoli, città e altro.
4. Veridicità, ovvero la validità dei dati: sono completi e accurati? Possiamo fidarci dei dati per prendere decisioni cruciali? Sono reali?

77. "A Very Short History Of Data Science." Accesso 1 novembre 2020. <https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/>.

78. "A Brief History of Data Analysis." Accesso 1 novembre 2020. <https://www.flydata.com/blog/a-brief-history-of-data-analysis/>.

79. Diebold, Francis. (2012). On the Origin(s) and Development of the Term "Big Data". SSRN Electronic Journal. 10.2139/ssrn.2152421. https://www.researchgate.net/publication/255967292_On_the_Origins_and_Development_of_the_Term_Big_Data.

80. "The Four V's of Big Data." Accesso 1 novembre 2020. <https://www.ibmbigdatahub.com/infographic/four-vs-big-data>.

81. Ci sono numerosi articoli e pubblicazioni che aggiungono molti altri termini a questo elenco di "V".

82. "Volume, velocity, and variety: Understanding the three V's of big data." Accesso 1 novembre 2020. <https://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/>.

83. "3Vs (volume, variety and velocity)." Accesso 1 novembre 2020. <https://whatis.techtarget.com/definition/3Vs>.

84. "Big Data: Forget Volume and Variety, Focus On Velocity." Accesso 1 novembre 2020. <https://www.forbes.com/sites/brentdykes/2017/06/28/big-data-forget-volume-and-variety-focus-on-velocity>.

La creazione digitale dei dati permette di averne una *varietà* di tipologie, in *volumi* straordinari e a *velocità* impressionanti. La legge di Moore e altre osservazioni analoghe ci hanno permesso di memorizzare i dati in maniera economica e di elaborarli e spostarli velocemente, il tutto con tassi di crescita esponenziali. La conservazione dei dati digitali è diventata così facile che ora possiamo comodamente ed economicamente mantenere *tutti* i dati che stiamo generando.⁸⁵ Tutto questo è big data.

La seguente citazione di Richard W. Hamming, sebbene sia del 1962, in qualche modo tratteggia la filosofia di questo libro:

*"Lo scopo dell'informatica è la comprensione, non i numeri."*⁸⁶

La data science sta producendo nuove conoscenze, più profonde e sottili, a un ritmo notevole. Sta davvero facendo la differenza. L'analisi dei big data è una parte fondamentale della risposta.

Per farvi un'idea della portata dei big data nell'industria, nel governo e nell'università, date un'occhiata a questa immagine ad alta risoluzione⁸⁷, che potete ingrandire per aumentarne la leggibilità:

http://mattturck.com/wp-content/uploads/2018/07/Matt_Turck_FirstMark_Big_Data_Landscape_2018_Final.png

1.13.2 Big data e data science stanno facendo la differenza: casi d'uso

La data science sta crescendo rapidamente perché produce importanti risultati che fanno la differenza. Elenchiamo casi d'uso di data science e big data nella tabella seguente. Ci aspettiamo che questi casi d'uso, insieme ai nostri esempi ed esercizi, siano fonte di ispirazione per interessanti progetti, studi, corsi, tesi e ricerche. L'analisi dei big data ha portato a maggiori profitti e migliori relazioni con i clienti, e ha anche permesso a squadre sportive di vincere più partite e campionati spendendo meno per i giocatori.^{88,89,90}

Casi d'uso di data science		
affidabilità del credito	immunoterapie	riconoscimento vocale
allocazione dinamica prezzi vendita	IoT e dispositivi medici	riduzione inquinamento
analisi del sentimento	mappatura cerebrale	rilevamento anomalie
analisi marketing	medicina di precisione	rilevamento emozioni
analisi del grafico sociale	medicina diagnostica	rilevamento frodi
assistenti intelligenti	medicina personalizzata	rilevamento malware
assistenti personali	medicina preventiva	rilevamento similarità
assistenza disabili	miglioramento sanità	rilevamento spam
auto a guida autonoma	minimizzazione rischi	rilevamento trend
cartelle sanitarie elettroniche	navigazione dinamica	risparmio energetico
classificazione della scrittura	nuovi farmaci	sequenziamento genoma umano
clienti: abbandono e fidelizzazione	predizione epidemie di malattie	servizi basati sulla posizione
clienti: agenti di servizio	predizione risultati di salute	sicurezza informatica



85. "How Much Information Is There In the World?" Accesso 1 novembre 2020. <http://www.lesk.com/mlesk/ksg97/ksg.html>. [Il seguente articolo ci ha indirizzato a quello di Michael Lesk: <https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/>.]

86. Hamming, R. W., *Numerical Methods for Scientists and Engineers* (New York: McGraw Hill, 1962). [Il seguente articolo ci ha indirizzato al libro di Hamming e alla relativa citazione: <https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/>.]

87. Turck, M., Hao, J., "Great Power, Great Responsibility: The 2018 Big Data & AI Landscape," <http://mattturck.com/bigdata2018/>.

88. Sawchik, T., *Big Data Baseball: Math, Miracles, and the End of a 20-Year Losing Streak* (New York: Flat Iron Books, 2015).

89. Ayres, I., *Super Crunchers* (Bantam Books, 2007), pp. 7-10.

90. Lewis, M., *Moneyball: The Art of Winning an Unfair Game* (W. W. Norton & Company, 2004).

 condivisione veicoli	predizione vendite basate sulle condizioni atmosferiche	sistemi di raccomandazione
consulenti finanziari automatici		smart city
contatori smart	prevenzione attacchi terroristici	smart home
controllo inventario	prevenzione furti	sottotitolazione automatica
controllo traffico	prevenzione furti d'identità	telemedicina
crimine: sorveglianza predittiva	prevenzione epidemie di malattie	termostati smart
diagnosi/trattamento tumori	previsioni del tempo	traduzione dei linguaggi
economia collaborativa	previsioni mercato azionario	traduzioni
editing genico	resa delle colture	valutazione appartamenti
eliminazione phishing	riassunti automatici	visione artificiale
estrazione dati	ricerca visuale prodotti	visualizzazione dati
gioco	riconoscimento facciale	

1.14 Caso pratico: un'applicazione mobile di big data

Nella vostra carriera, lavorerete con molti linguaggi di programmazione e tecnologie software. Con 130 milioni di utenti mensili attivi,⁹¹ Waze di Google è una delle applicazioni di big data più utilizzate per la navigazione GPS. I primi navigatori GPS si basavano su mappe statiche e coordinate GPS per determinare la strada migliore, e non potevano cambiare dinamicamente per adattarsi al traffico.

Waze elabora un'enorme quantità di dati ottenuti via **crowdsourcing**, ovvero attraverso dati forniti in tempo reale dagli utenti e dai loro dispositivi. Attraverso l'analisi di questi dati, Waze determina la miglior strada per arrivare a una destinazione nel minor tempo possibile. Per farlo, Waze si affida alla connessione a Internet del vostro cellulare. L'applicazione invia ai propri server aggiornamenti automatici sulla posizione attuale (previo vostro consenso), e questi dati vengono usati per cambiare dinamicamente la strada in base alle condizioni di traffico e per calibrare meglio le mappe. Gli utenti riportano anche altre informazioni, come blocchi stradali, ostacoli, lavori in corso, veicoli fermi, presenza della polizia, prezzi della benzina e molto altro. Waze si occupa poi di avvisare gli altri conducenti presenti in zona.

Waze utilizza diverse tecnologie per fornire questi servizi. Non siamo al corrente di come sia implementata Waze, ma possiamo dedurre alcune delle tecnologie che probabilmente utilizza, tra cui le seguenti.

- Molte app utilizzano almeno un software open source. Sfrutteremo diverse librerie e strumenti open source nei casi pratici.
- Waze scambia informazioni tra i suoi server e i dispositivi mobile dei suoi utenti attraverso Internet. Solitamente questi dati sono scritti nel formato JSON (*JavaScript Object Notation*). Molte librerie usano questo formato senza mostrarlo al programmatore.
- Waze utilizza la sintesi vocale per avvisare i conducenti e comunicare le direzioni di guida; utilizza inoltre il riconoscimento vocale per interpretare i comandi vocali. Molti fornitori di cloud offrono funzionalità di sintesi vocale e riconoscimento vocale.
- Dopo aver convertito un comando vocale in testo, Waze determina la corretta azione da svolgere, il che richiede l'elaborazione del linguaggio naturale (NLP).
- Waze aggiorna dinamicamente le visualizzazioni, come segnali di allerta e mappe interattive.
- Waze utilizza i cellulari come se fossero dispositivi IoT per ottenere un flusso di dati. Ogni cellulare è un sensore GPS che trasmette continuamente dati a Waze attraverso Internet.
- Waze riceve dati IoT da milioni di cellulari in ogni istante. Deve elaborarli, memorizzarli e analizzarli immediatamente per aggiornare le mappe dei dispositivi, per mostrare e comunicare messaggi di allerta o possibili cambi di direzione. Tutto questo richiede enormi capacità di calcolo parallelo implementate con

91. "Waze Communities," Accesso 1 novembre 2020. <https://www.waze.com/communities>.

agglomerati di computer nel cloud. Si possono usare diverse infrastrutture tecnologiche per la ricezione di flussi di dati, per la memorizzazione di questi big data in database appropriati e per l'elaborazione con software e hardware che mettono a disposizione enormi capacità di calcolo parallelo.

- Waze utilizza tecniche di intelligenza artificiale per analizzare i dati, e questo permette di prevedere le strade migliori sulla base delle informazioni ricevute. Potete utilizzare il machine learning per analizzare enormi quantità di dati e il deep learning per ottenere previsioni basate su di essi.
- Probabilmente Waze utilizza un database a grafo per memorizzare le informazioni dei percorsi. Con questi database si può calcolare efficientemente la strada più corta. Potete utilizzare database grafici, come Neo4J.
- Molte auto moderne sono equipaggiate con dispositivi che permettono loro di “vedere” ostacoli o altre auto in prossimità. Questi dispositivi si usano per implementare sistemi di frenata automatica e sono ingredienti base nelle tecnologie delle macchine a guida autonoma. Piuttosto che affidarsi agli utenti per avere informazioni su ostacoli o macchine ferme sul lato della strada, le app di navigazione potrebbero sfruttare videocamere o altri sensori usando tecniche di visione artificiale basate su deep learning per analizzare immagini in tempo reale e segnalare automaticamente questi oggetti. Potete ricorrere al deep learning per la visione artificiale.

1.15 Intelligenza artificiale, all'intersezione tra informatica e data science

Quando un neonato apre gli occhi per la prima volta “vede” le facce dei suoi genitori? Capisce che cos’è una faccia? O anche solo una forma? I neonati devono imparare a conoscere il mondo che li circonda. Questo è quello che l’intelligenza artificiale (IA) sta facendo oggi: sta guardando enormi quantità di dati e imparando da essi. L’IA viene usata per giocare, implementare applicazioni di visione artificiale, permettere la guida autonoma nelle auto, permettere ai robot di imparare nuove azioni, diagnosticare condizioni mediche, fare traduzioni in tempo reale, creare chatbot che rispondono a domande arbitrarie usando enormi database, ecc. Chi avrebbe potuto immaginare solo qualche anno fa che automobili intelligenti a guida autonoma avrebbero viaggiato sulle nostre strade? Eppure, questo è un ambito altamente concorrenziale. L’obiettivo finale di tutto questo apprendimento è l’**intelligenza artificiale generale**, ovvero una IA capace di effettuare compiti intelligenti allo stesso modo di un essere umano.

Pietre miliari dell’intelligenza artificiale

Alcuni successi dell’intelligenza artificiale hanno catturato l’attenzione e l’immaginario dell’opinione pubblica, facendo capire alla gente che l’IA è una cosa reale e alle aziende che conviene commercializzarla. Eccone alcuni.

- Nella partita del 1997 tra il sistema **DeepBlue di IBM** e il grande maestro internazionale di scacchi Gary Kasparov, DeepBlue divenne il primo computer a battere un campione mondiale di scacchi in carica usando regole da campionato.⁹² IBM aveva caricato in DeepBlue centinaia di migliaia di partite di scacchi tra campioni. DeepBlue utilizzò algoritmi di *forza bruta* per valutare più di 200 milioni di mosse al secondo!⁹³ Questo è il lavoro dei big data. IBM ricevette il Carnegie Mellon University Fredkin Prize, 100.000 dollari in palio dal 1980 per il creatore del primo computer in grado di battere un campione di scacchi.⁹⁴
- Nel 2011, **Watson di IBM** batté i due migliori giocatori di Jeopardy! in una sfida da 1 milione di dollari. Watson usò simultaneamente centinaia di tecniche per l’analisi del linguaggio per rintracciare le risposte corrette tra 200 milioni di pagine (che includevano tutta Wikipedia) che hanno richiesto

92. “Deep Blue versus Garry Kasparov.” Accesso 1 novembre 2020. https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov.

93. “Deep Blue (chess computer).” Accesso 1 novembre 2020. [https://en.wikipedia.org/wiki/Deep_Blue_\(chess_computer\)](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)).

94. “IBM Deep Blue Team Gets \$100,000 Prize.” Accesso 1 novembre 2020. <https://articles.latimes.com/1997/jul/30/news/mn-17696>.

quattro terabyte di spazio di memorizzazione.^{95,96} L'allenamento di Watson fu condotto con l'utilizzo di tecniche di machine-learning e di apprendimento con rinforzo.⁹⁷ Potenti librerie consentono di eseguire machine-learning e apprendimento con rinforzo in vari linguaggi di programmazione.

- Go – un gioco da tavolo inventato in Cina migliaia di anni fa⁹⁸ – è considerato tra i giochi più complessi mai inventati, con 10^{170} configurazioni possibili.⁹⁹ Per darvi un'idea di quanto è grande questo numero, si crede che nell'universo conosciuto ci siano (solo) tra 10^{78} e 10^{82} atomi!^{100,101} Nel 2015, **AlphaGo**, creato dal gruppo DeepMind di Google, utilizzò un *algoritmo di deep learning con due reti neurali per battere il campione europeo di Go Fan Hui*. Go è considerato molto più complesso degli scacchi. Potenti librerie consentono di usare reti neurali per il deep learning.
- Più recentemente Google ha generalizzato l'IA di AlphaGo creando **AlphaZero**, una IA che *apprende da sola come giocare a nuovi giochi*. Nel dicembre 2017, AlphaZero imparò le regole degli scacchi apprendendo da sola come giocare in meno di quattro ore usando l'apprendimento con rinforzo. Dopotutto, batté il programma campione mondiale di scacchi, Stockfish 8, in una sfida di 100 partite, vincendo o pareggiando ogni partita. Dopo essersi allenato a Go per solo otto ore, AlphaZero fu in grado di giocare a Go contro AlphaGo vincendo 60 partite su 100.¹⁰²

IA: un settore con problemi ma senza soluzioni

Per molte decadi, l'IA è stata un settore che poneva problemi ma senza soluzioni. Questo perché, una volta che si trova la soluzione a un particolare problema, la gente dice “Beh, questa non è intelligenza, è solo un programma che dice al computer esattamente cosa fare”. Tuttavia, usando il machine learning, il deep learning e l'apprendimento con rinforzo, non stiamo pre-programmando soluzioni a *specifici* problemi, stiamo invece permettendo ai nostri computer di risolvere i problemi imparando dai dati, e solitamente da grandi quantità di essi. Molti dei problemi più interessanti e impegnativi vengono affrontati con il deep learning. Solo Google sta sviluppando migliaia di progetti di deep learning.^{103,104}

✓ Autovalutazione

1. (*Completare*) L'obiettivo finale dell'IA è produrre una _____.

Risposta: intelligenza artificiale generale.

2. (*Completare*) Watson di IBM batté i due migliori giocatori di Jeopardy! L'allenamento di Watson fu condotto usando una combinazione di tecniche di _____ learning e di _____ learning.

Risposta: machine, reinforcement.

3. (*Completare*) _____ di Google imparò le regole degli scacchi apprendendo da sola come giocare in meno di quattro ore, usando l'apprendimento con rinforzo. Dopotutto batté il programma campione mondiale di scacchi, Stockfish 8, in una sfida di 100 partite, vincendo o pareggiando ogni partita.

Risposta: AlphaZero.

95. “IBM Watson: The inside story of how the Jeopardy-winning supercomputer was born, and what it wants to do next.” Accesso 1 novembre 2020. <https://www.techrepublic.com/article/ibm-watson-the-inside-story-of-how-the-jeopardy-winning-supercomputer-was-born-and-what-it-wants-to-do-next/>.

96. “Watson (computer).” Accesso 1 novembre 2020. [https://en.wikipedia.org/wiki/Watson_\(computer\)](https://en.wikipedia.org/wiki/Watson_(computer)).

97. “Building Watson: An Overview of the DeepQA Project.” Accesso 1 novembre 2020. <https://www.aaai.org/Magazine/Watson/watson.php, AI Magazine, Fall 2010>.

98. “A Brief History of Go.” Accesso 1 novembre 2020. <http://www.usgo.org/brief-history-go>.

99. “Google artificial intelligence beats champion at world's most complicated board game.” Accesso 1 novembre 2020. <https://www.pbs.org/newshour/science/google-artificial-intelligence-beats-champion-at-worlds-most-complicated-board-game>.

100. “How Many Atoms Are There in the Universe?” Accesso 1 novembre 2020. <https://www.universetoday.com/36302/atoms-in-the-universe/>.

101. “Observable universe.” Accesso 1 novembre 2020. https://en.wikipedia.org/wiki/Observable_universe#Matter_content.

102. “AlphaZero AI beats champion chess program after teaching itself in four hours.” Accesso 1 novembre 2020. <https://www.theguardian.com/technology/2017/dec/07/alphazero-google-deepmind-ai-beats-champion-program-teaching-itself-to-play-four-hours>.

103. “Google has more than 1,000 artificial intelligence projects in the works.” Accesso 1 novembre 2020. <http://theweek.com/speedreads/654463/google-more-than-1000-artificial-intelligence-projects-works>.

104. “Google says ‘exponential’ growth of AI is changing nature of compute.” Accesso 1 novembre 2020. <https://www.zdnet.com/article/google-says-exponential-growth-of-ai-is-changing-nature-of-compute/>.

1.16 Riepilogo

Esercizi di autovalutazione

- 1.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
- I computer elaborano i dati sotto il controllo di sequenze di istruzioni chiamate _____.
 - Le unità logiche fondamentali del computer sono: _____, _____, _____, _____, e _____.
 - I tre tipi di linguaggi di programmazione trattati nel capitolo sono _____, _____ e _____.
 - I programmi che traducono nel linguaggio macchina programmi in un linguaggio di alto livello sono chiamati _____.
 - _____ è un sistema operativo per dispositivi mobili basato sul kernel Linux.
 - Un _____ consente a un dispositivo di rispondere al movimento.
 - Il C è diffusamente noto come il linguaggio di sviluppo del sistema operativo _____.
- 1.2 Riempite gli spazi vuoti in ognuna delle seguenti frasi riguardanti l'ambiente C.
- I programmi in C sono normalmente scritti su un computer usando un _____.
 - In un sistema in C, un _____ viene eseguito automaticamente prima che inizi la fase di traduzione.
 - Il _____ combina l'output del compilatore con varie funzioni di libreria per produrre un'immagine eseguibile.
 - Il _____ trasferisce l'immagine eseguibile dal disco alla memoria.

Risposte agli esercizi di autovalutazione

- 1.1 a) programmi. b) unità di input, unità di output, unità di memoria, unità di elaborazione centrale, unità aritmetica e logica, unità di memoria secondaria. c) linguaggi macchina, linguaggi assembly, linguaggi di alto livello. d) compilatori. e) Android. f) accelerometro. g) UNIX.
- 1.2 a) editor. b) preprocessore. c) linker. d) loader.

Esercizi

- 1.3 Classificate ognuno dei seguenti elementi o come hardware o come software.
- CPU
 - compilatore C
 - ALU
 - preprocessore C
 - unità di input
 - un programma editor
- 1.4 (*Organizzazione del computer*) Riempite gli spazi vuoti in ognuna delle seguenti frasi.
- L'unità logica che riceve informazioni dall'esterno del computer per l'utilizzo da parte del computer si chiama _____.
 - _____ è l'unità logica che invia informazioni già elaborate dal computer a diversi dispositivi in modo che possano essere utilizzate all'esterno del computer.
 - _____ e _____ sono unità logiche del computer che conservano le informazioni.
 - _____ è l'unità logica del computer che esegue i calcoli.
 - _____ è l'unità logica del computer preposta a prendere decisioni logiche.
 - La _____ è un'unità logica del computer che coordina le attività di tutte le altre unità logiche.
- 1.5 Analizzate il significato di ognuno dei seguenti nomi.
- `stdin`
 - `stdout`
 - `stderr`

1.6 (Neutralità di genere) Scrivete in inglese i passi di una procedura manuale che analizzi un paragrafo di testo e rimpiazzi tutte le parole che si riferiscono a un genere specifico con parole che non hanno connotazione di genere. Assumete che vi sia una lista di parole da sostituire con un rimpiazzo adatto (per esempio, rimpiazzare "wife" o "husband" con "spouse", "man" o "woman" con "person", "daughter" o "son" con "child" e così via), spiegate la procedura che usereste per leggere il paragrafo di testo ed eseguire manualmente questi rimpiazzi. In che modo la procedura potrebbe generare uno strano termine come "woperchild" e come potreste modificarla per evitare questa possibilità? Nel Capitolo 3 imparerete che "algoritmo" è il termine formale per "procedura", e che un algoritmo specifica i passi da compiere e l'ordine in cui vanno eseguiti.

1.7 (Automobili a guida autonoma) Solo qualche anno fa pensare a macchine senza conducente sulle nostre strade sarebbe stato impossibile (infatti i controllori ortografici d'inglese non riconoscono la parola "driverless", ovvero "senza autista"). Queste auto sono rese possibili da molte delle tecnologie che vedrete in questo libro e sono già comuni in alcuni luoghi.

- Se chiamando un taxi si fermasse una macchina senza conducente, vi salireste? Sareste a vostro agio nel dirle la vostra destinazione e credereste che vi ci possa portare? Che misure di sicurezza vorreste a bordo? Cosa fareste se la macchina partisse nella direzione sbagliata?
- Cosa succederebbe se due auto senza conducente entrassero in una strada a senso unico dalle due direzioni opposte? Che protocollo dovrebbero seguire per determinare quale auto deve procedere?
- Immaginate di trovarvi dietro a una macchina in coda a un semaforo rosso. Il semaforo diventa verde e la macchina non si muove, voi suonate il clacson ma non succede niente. Allora scendete e vedete che l'auto è senza conducente. Cosa fareste?
- Se un poliziotto fermasse un'auto senza conducente per eccesso di velocità con voi a bordo, chi o cosa dovrebbe pagare la multa?
- Una preoccupazione riguardante le auto senza conducente è che potrebbero essere violate informaticamente. Qualcuno potrebbe pericolosamente impostare un'alta (o bassa) velocità. Cosa accadrebbe se vi indirizzassero a una destinazione diversa da quella che volete voi?

1.8 (Ricerca: riproducibilità) Un concetto cruciale negli studi di data science è la riproducibilità, la possibilità di riprodurre i risultati per sé e per gli altri. Fate una ricerca sulla riproducibilità ed elencate i concetti usati per creare risultati riproducibili negli studi di data science.

1.9 (Ricerca: intelligenza artificiale generale) Uno degli obiettivi più ambiziosi nel campo dell'IA è il raggiungimento dell'*intelligenza artificiale generale*, il punto in cui le macchine saranno intelligenti tanto quanto gli esseri umani. Fate una ricerca su questo interessante argomento. Quando accadrà? Quali i problemi etici che andranno affrontati? L'intelligenza umana sembra essere stabile su periodi molto lunghi. Computer potenti dotati di intelligenza artificiale generale potrebbero plausibilmente (e velocemente) evolvere ben oltre l'intelligenza umana. Discutete i problemi che questo comporta.

1.10 (Ricerca: assistenti intelligenti) Molte aziende oggi offrono assistenti digitali intelligenti, come Watson di IBM, Alexa di Amazon, Siri di Apple, l'Assistente Google e Cortana di Microsoft. Fate una ricerca su questi e altri sistemi, elencandone gli utilizzi che possono migliorare la vita delle persone. Fate una ricerca sui problemi di etica e di privacy relativi agli assistenti intelligenti. Trovate aneddoti divertenti sugli assistenti intelligenti.

1.11 (Ricerca: IA nella sanità) Fate una ricerca sul settore in rapida crescita delle applicazioni IA nella sanità. Per esempio, supponete che un programma di diagnostica abbia accesso a tutte le lastre fatte finora e alle diagnosi associate (si tratta sicuramente di un esempio di big data). Le applicazioni di visione artificiale integrate con il deep learning possono essere impiegate per elaborare questi dati "etichettati" per imparare a diagnosticare problemi medici. Fate una ricerca sul deep learning nella medicina diagnostica e descrivete i suoi risultati più importanti. Quali sono i problemi etici dell'avere macchine e non dottori umani che effettuano diagnosi mediche? Vi fidereste di una diagnosi effettuata da una macchina? Chiedereste una seconda opinione?

1.12 (Ricerca: normative su privacy e integrità dei dati) Nella Prefazione abbiamo citato la legge sulla portabilità e responsabilità delle polizze di assicurazione sanitaria (HIPAA, *Health Insurance Portability and Accountability Act*) e la legge sulla privacy dei consumatori in California (CCPA, *California Consumer Privacy Act*) per quanto riguarda gli Stati Uniti, e il regolamento generale per la protezione dei dati nell'Unione Eu-

ropea (GDPR, *General Data Protection Regulation*). Normative di questo tipo stanno diventando sempre più diffuse e rigorose. Esaminate ciascuna di queste leggi e i loro effetti sulla vostra privacy.

1.13 (Ricerca: informazioni di identificazione personale) La protezione delle informazioni di identificazione personale (PII, *Personal Identifiable Information*) è un aspetto fondamentale della privacy. Fate una ricerca su questo argomento ed esponete i vostri commenti.

1.14 (Ricerca: big data, IA e cloud; come le aziende usano queste tecnologie) Scegliete un'organizzazione importante e fate una ricerca su come potrebbe usare ognuna delle seguenti tecnologie: IA, big data, il cloud, applicazioni mobile, elaborazione linguaggio naturale, riconoscimento vocale, sintesi vocale, database, machine learning, deep learning, apprendimento con rinforzo, Hadoop, Spark, IoT e servizi web.

1.15 (Ricerca: Raspberry Pi e l'Internet delle cose) Oggigiorno è possibile avere un computer nel cuore di ogni dispositivo, per poi connettere questi dispositivi a Internet. Tutto questo ha portato alla creazione dell'Internet delle cose che già ora connette decine di miliardi di dispositivi. La scheda Raspberry Pi è un computer economico che spesso è nel nucleo di molti dispositivi IoT. Fate una ricerca sulla Raspberry Pi e su qualche applicazione IoT in cui è usata.

1.16 (Ricerca: etica dei deep fake) Le tecniche di IA hanno reso possibile la creazione di *deep fake* permettendo di catturare le fattezze, la voce, i movimenti e le espressioni facciali delle persone per imporli poi su video falsi ma molto realistici. È possibile realizzarli in modo che nel video si dica o si faccia qualsiasi cosa. Fate una ricerca sui problemi etici dei deep fake. Cosa fareste se accendendo la TV apparisse un video deep fake in cui un ufficiale del governo o un giornalista annunciasse un imminente attacco nucleare? Provate a cercare informazioni su Orson Welles e sul suo programma radiofonico "War of the Worlds" ("La guerra dei mondi") che scatenò il panico nel 1938.

1.17 (Ricerca: Blockchain, un mondo di opportunità) Le criptovalute come Bitcoin ed Ethereum sono basate su una tecnologia chiamata blockchain che ha visto una crescita esplosiva negli ultimi anni. Fate una ricerca sulle origini della blockchain, sulle sue applicazioni e su come è usata per implementare le criptovalute. Cercate anche altre applicazioni di questa tecnologia. Nei prossimi anni ci saranno molte straordinarie opportunità per gli sviluppatori software con esperienza nella realizzazione di applicazioni basate su blockchain.

1.18 (Ricerca: la programmazione sicura in C e la CERT Division del Software Engineering Institute della Carnegie Mellon University) L'esperienza ha dimostrato come sia molto difficile costruire sistemi informatici a livello industriale in grado di resistere agli attacchi, che possono essere istantaneei e di portata globale. I sistemi informatici di molte delle più grandi aziende, agenzie governative e organizzazioni militari sono stati compromessi. Le vulnerabilità del software derivano spesso da semplici problemi di programmazione. Includere aspetti relativi alla sicurezza nel software dall'inizio del ciclo di sviluppo può ridurre sensibilmente le vulnerabilità. Il Software Engineering Institute (SEI) della Carnegie Mellon University ha creato la divisione CERT (<https://www.sei.cmu.edu/about/divisions/cert/index.cfm>) per analizzare e rispondere prontamente agli attacchi. CERT pubblica e promuove standard di codifica sicura per aiutare i programmati in C (e non solo) a implementare sistemi di portata industriale evitando pratiche di programmazione che espongono i sistemi agli attacchi. Gli standard CERT si evolvono quando sorgono nuovi problemi di sicurezza. L'obiettivo dello standard di codifica SEI CERT per il C è di "rinforzare" i sistemi informatici e le applicazioni in modo che resistano agli attacchi. Fate una ricerca sul CERT e discutetene i risultati raggiunti e le nuove sfide da affrontare. Per aiutarvi a mettere a punto le pratiche di programmazione sicura in C, i Capitoli 2-12 e 14 includono paragrafi su questo specifico argomento, che trattano alcuni temi e tecniche fondamentali oltre a fornire link e riferimenti per un approfondimento ulteriore.

1.19 (Ricerca: Watson di IBM) IBM è partner di decine di migliaia di aziende che spaziano in tantissimi ambiti, incluso il nostro editore Pearson. Fate una ricerca sui risultati chiave ottenuti da Watson e sul tipo di sfide che IBM e i suoi partner stanno affrontando.

CAPITOLO

2

Sommario del capitolo

- 2.1 Introduzione
- 2.2 Un semplice programma in C: stampare una riga di testo
- 2.3 Un altro semplice programma in C: sommare due interi
- 2.4 Concetti relativi alla memoria
- 2.5 Arithmetica in C
- 2.6 Decisioni: operatori di uguaglianza e relazionali
- 2.7 Programmazione sicura in C
- 2.8 Riepilogo

Introduzione alla programmazione nel linguaggio C

Obiettivi

- Scrivere semplici programmi in C.
- Usare semplici istruzioni di input e output.
- Usare i tipi di dati fondamentali.
- Apprendere i concetti relativi alla memoria di un computer.
- Usare operatori aritmetici.
- Imparare l'ordine di precedenza degli operatori aritmetici.
- Scrivere semplici istruzioni per prendere decisioni.
- Introdurre alle pratiche per una programmazione sicura in C.

2.1 Introduzione

Il linguaggio C facilita un approccio strutturato e disciplinato alla progettazione di programmi per computer. In questo capitolo introdurremo la programmazione in C e presenteremo diversi esempi che illustrano molte caratteristiche importanti del C. Ciascun esempio verrà analizzato un'istruzione per volta. Nei Capitoli 3 e 4 presenteremo un'introduzione alla programmazione strutturata in C, una metodologia che vi aiuterà a produrre programmi chiari e di facile manutenzione. Useremo poi l'approccio strutturato per tutto il resto del testo. Questo capitolo si conclude con il primo dei nostri paragrafi “Programmazione sicura in C”.

2.2 Un semplice programma in C: stampare una riga di testo

Iniziamo con un semplice programma in C che stampa una riga di testo. Il programma e il suo output sullo schermo sono mostrati nella Figura 2.1.

```
1 // fig02_01.c
2 // Un primo programma in C.
3 #include <stdio.h>
4
5 // inizio dell'esecuzione del programma con main
6 int main(void) {
7     printf("Welcome to C!\n");
8 } // fine della funzione main
```

Welcome to C!

Figura 2.1 Un primo programma in C.

Commenti

Le righe 1 e 2

```
// fig02_01.c
// Un primo programma in C.
```

iniziano con `//`, per indicare che queste due righe sono **commenti**. Inserite i commenti per **documentare i programmi** e migliorarne la leggibilità. I commenti non portano il computer a compiere alcuna azione durante l'esecuzione del programma: vengono semplicemente ignorati. È nostra convenzione in ogni programma utilizzare il commento della riga 1 per specificare il nome del file, e il commento della riga 2 per descrivere l'obiettivo del programma. I commenti aiutano anche altre persone a leggere e a capire il vostro programma.

- ⊗ Potete usare anche **commenti multilinea** `/*...*/` in cui ogni cosa compresa tra `/*` nella prima riga e `*/` alla fine dell'ultima è un commento. Sono preferibili commenti con `//` perché sono più brevi ed eliminano gli errori comuni di programmazione che possono capitare con commenti `/*...*/`, come quando si omette accidentalmente la chiusura `*/`.

Direttiva #include per il preprocessore

La riga 3

```
#include <stdio.h>
```

è una **direttiva per il preprocessore C**. Le righe che cominciano con `#` sono elaborate dal preprocessore prima della compilazione. La riga 3 dice al preprocessore di includere i contenuti del **file di intestazione** (header) **di input/output standard** (`<stdio.h>`). Il file di intestazione contiene informazioni utilizzate dal compilatore per garantire un corretto utilizzo delle funzioni della libreria di input/output standard come `printf` (riga 7). Spiegheremo più dettagliatamente i contenuti dei file di intestazione nel Capitolo 5.

Righe vuote e caratteri di spaziatura

La riga 4 è semplicemente una riga vuota. Usate righe vuote e i caratteri di spazio e di tabulazione (cioè "tab") per rendere i programmi più leggibili. Questi caratteri sono noti come **caratteri di spaziatura** e sono normalmente ignorati dal compilatore.

La funzione main

La riga 6

```
int main(void) {
```

è parte di ogni programma in C. Le parentesi dopo `main` indicano che `main` è un blocco costituente di un programma chiamato **funzione**. I programmi in C contengono una o più funzioni, una delle quali deve essere proprio `main`. Ciascun programma inizia l'esecuzione dalla funzione `main`. Una buona regola è far precedere ogni funzione con un commento (come nella riga 5) che ne indichi lo scopo.

Le funzioni possono restituire informazioni. La parola chiave `int` alla sinistra di `main` indica che `main` "restituisce" un valore intero (numero intero). Spiegheremo cosa significa per una funzione "restituire un valore" nel Capitolo 4 quando useremo una funzione matematica per eseguire un calcolo e nel Capitolo 5 quando vi mostreremo come creare le vostre funzioni. Per ora include semplicemente la parola chiave `int` alla sinistra di `main` in ognuno dei vostri programmi.

Le funzioni possono anche ricevere informazioni quando vengono eseguite. Il `void` qui tra parentesi vuol dire che `main` non riceve alcuna informazione. Nel Capitolo 15 mostreremo un esempio di `main` che riceve informazioni.

Una parentesi graffa sinistra, `{`, inizia il **corpo** di ogni funzione (riga 6). Una corrispondente parentesi graffa destra, `}`, termina ogni funzione (riga 8). Quando un programma raggiunge la parentesi graffa destra di chiusura di `main` termina l'esecuzione. Questa coppia di parentesi graffe e la porzione del programma fra le parentesi è chiamata **blocco**, un importante costrutto sintattico che analizzeremo più approfonditamente nei capitoli successivi.

Un'istruzione di output

La riga 7

```
printf("Welcome to C!\n");
```

fa compiere un'azione al computer, ossia fa stampare sullo schermo la **stringa** di caratteri compresa tra le virgolette. Una stringa è talvolta chiamata **stringa di caratteri, messaggio o letterale**.

L'intera riga 7, comprendente la "chiamata" alla funzione printf per svolgere il suo compito, l'**argomento** di printf tra le parentesi e il punto e virgola (;), è chiamata **istruzione**. Ogni istruzione deve terminare con un punto e virgola (noto anche come **terminatore dell'istruzione**). La "f" di printf sta per "formattato". Quando la riga 7 viene eseguita, essa stampa sullo schermo il messaggio Welcome to C!. I caratteri, di norma, vengono stampati così come appaiono tra le doppie virgolette, salvo i caratteri \n che non vengono visualizzati.

Sequenze di escape

Il carattere di backslash (\) contenuto in una stringa è detto **carattere di escape**. Esso indica che printf è tenuta a fare qualcosa fuori dall'ordinario. In una stringa, il compilatore combina un backslash con il carattere successivo per formare una **sequenza di escape**. La sequenza di escape \n è detta **newline** (nuova riga). Quando printf incontra il carattere di nuova riga in una stringa, il cursore viene posizionato all'inizio della riga successiva sullo schermo. Di seguito sono elencate alcune comuni sequenze di escape.

Sequenza di escape	Descrizione
\n	Posiziona il cursore all'inizio della riga successiva.
\t	Sposta il cursore alla successiva tabulazione orizzontale.
\a	Produce un suono o un allarme visibile senza cambiare la posizione corrente del cursore.
\\\	Poiché il backslash ha un significato speciale in una stringa, viene richiesto \\ per inserire un carattere di backslash in una stringa.
\\"	Poiché le stringhe sono racchiuse tra doppie virgolette, viene richiesto \\ per inserire un carattere di doppie virgolette in una stringa.

Il linker e gli eseguibili

Funzioni della Libreria Standard come printf e scanf non sono parti del linguaggio di programmazione C. Per esempio, il compilatore non è in grado di trovare un errore di ortografia in printf o scanf. Durante la compilazione di un'istruzione printf, il compilatore introduce semplicemente uno spazio nel programma oggetto per una "chiamata" alla funzione della libreria. Ma il compilatore non sa dove sono le funzioni della libreria, mentre chi lo sa è il linker. Quando il linker è in esecuzione, localizza le funzioni della libreria e inserisce le chiamate appropriate a queste funzioni nel programma oggetto. A questo punto il programma oggetto è completo e pronto per essere eseguito. Il programma elaborato dal linker è chiamato **eseguibile**. Se il nome della funzione è scritto male, il linker scoprirà l'errore, perché non sarà in grado di collegare il nome nel programma in C con il nome di una qualsiasi funzione conosciuta nelle librerie.



Convenzioni per l'indentazione

Fate rientrare a destra il testo dell'intero corpo di ciascuna funzione con un livello di indentazione (raccomandiamo tre spazi) all'interno delle parentesi graffe che lo delimitano. Questa indentazione mette in risalto la struttura funzionale dei programmi e aiuta a renderli più leggibili.

Stabilite una convenzione per l'entità dell'indentazione che preferite e poi applicatela uniformemente. Il tasto Tab può essere utilizzato per effettuare le indentazioni, ma le tabulazioni possono variare. Le guide stilistiche professionali spesso raccomandano di utilizzare spazi anziché tabulazioni. Alcuni editor di codice inseriscono spazi in risposta alla pressione del tasto Tab.

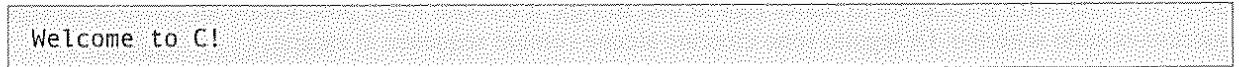
Uso di diversi printf

La funzione printf è in grado di stampare Welcome to C! in tanti modi diversi. Per esempio, il programma della Figura 2.2 usa due istruzioni per produrre lo stesso output della Figura 2.1. Questo funziona perché ogni printf riprende a stampare da dove la precedente ha completato la stampa. La riga 7 stampa Welcome seguito da uno spazio (ma senza un carattere di newline); la funzione printf della riga 8 inizia a stampare sulla stessa riga immediatamente dopo lo spazio.

```

1 // fig02_02.c
2 // Stampare su una riga con due istruzioni printf.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     printf("Welcome ");
8     printf("to C!\n");
9 } // fine della funzione main

```



Welcome to C!

Figura 2.2 Stampare su una riga con due istruzioni printf.

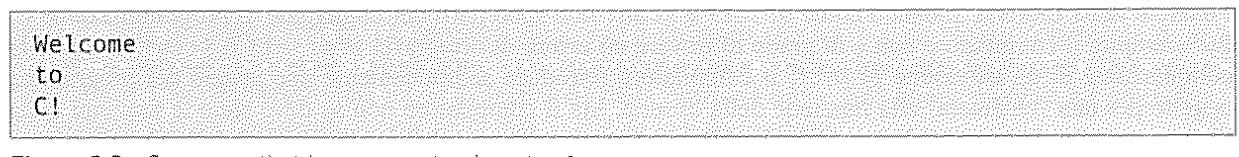
Stampare più righe con una singola printf

Una printf può stampare diverse righe, come mostrato nella Figura 2.3. Ogni \n sposta il cursore dell'output all'inizio della riga successiva.

```

1 // fig02_03.c
2 // Stampare diverse righe con una singola printf.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     printf("Welcome\n");
8     printf("to\n");
9     printf("C!");
10 }

```



Welcome
to
C!

Figura 2.3 Stampare più righe con una singola printf.

✓ Autovalutazione

1. (*Scelta multipla*) Considerate il codice:

```
int main(void)
```

Quale delle seguenti affermazioni è falsa?

- a) Le parentesi dopo main indicano che si tratta di una funzione.
- b) La parola chiave int a sinistra di main indica che main restituisce un valore intero, e il void tra parentesi indica che main non riceve alcuna informazione.
- c) Una parentesi tonda sinistra, (, inizia il corpo di ogni funzione. Una corrispondente parentesi tonda destra,), termina il corpo di ogni funzione.
- d) Quando l'esecuzione raggiunge la fine di main, il programma termina.

Risposta: c) è *falsa*. In realtà, una parentesi graffa sinistra, {, inizia il corpo di ogni funzione, e una corrispondente parentesi graffa destra, }, termina il corpo di ogni funzione.

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) Ogni printf riprende a stampare da dove la printf precedente ha completato la stampa.
- b) Nel codice seguente, la prima printf stampa Welcome seguito da uno spazio, e la seconda printf inizia a stampare sulla riga di output successiva:

```
printf("Welcome ");
printf("to C!\n");
```

- c) La seguente printf stampa diverse righe di testo:

```
printf("Welcome\nTo\nC!\n");
```

- d) Ogni volta che si incontra la sequenza di escape \n, l'output continua all'inizio della riga successiva.

Risposta: b) è *falsa*. In realtà, la seconda printf inizia a stampare immediatamente dopo lo spazio inviato in uscita dalla prima printf.

2.3 Un altro semplice programma in C: sommare due interi

Il nostro programma successivo usa la funzione scanf della Libreria Standard per ricevere due valori interi scritti da un utente sulla tastiera, poi ne calcola la somma e stampa il risultato usando printf. Il programma e un esempio di output sono mostrati nella Figura 2.4. Nel riquadro di input/output della Figura 2.4 evidenziamo in **grassetto** i numeri digitati dall'utente.

```
1 // fig02_04.c
2 // Programma per la somma.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     int integer1 = 0; // conterrà il primo numero inserito dall'utente
8     int integer2 = 0; // conterrà il secondo numero inserito dall'utente
9
10    printf("Enter first integer: "); // prompt
11    scanf("%d", &integer1); // legge un intero
12
13    printf("Enter second integer: "); // prompt
14    scanf("%d", &integer2); // legge un intero
15
16    int sum = 0; // variabile nella quale viene memorizzata la somma
17    sum = integer1 + integer2; // assegna il totale a sum
18
19    printf("Sum is %d\n", sum); // stampa la somma
20 } // fine della funzione main
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Figura 2.4 Programma per la somma.

Il commento nella riga 2 spiega l'obiettivo del programma. Anche in questo caso, il programma inizia l'esecuzione con `main` (righe 6-20); le parentesi graffe alle righe 6 e 20 segnano rispettivamente l'inizio e la fine del corpo di `main`.

Variabili e definizioni di variabili

Le righe 7 e 8

```
int integer1 = 0; // conterrà il primo numero inserito dall'utente
int integer2 = 0; // conterrà il secondo numero inserito dall'utente
```

sono **definizioni**. I nomi `integer1` e `integer2` si riferiscono a **variabili**, locazioni in memoria dove si possono memorizzare i valori che verranno utilizzati dal programma. Queste definizioni specificano che le variabili `integer1` e `integer2` sono di tipo `int`, il che significa che avranno valori interi, come 7, -11, 0 e 31914. Le righe 7 e 8 **inizializzano** ogni variabile a 0 facendo seguire il nome della variabile con un `=` e un valore. Sebbene non sia necessario inizializzare in modo esplicito ogni variabile, ciò consentirà di evitare molti problemi comuni.

Definire le variabili prima che siano utilizzate

Tutte le variabili devono essere definite con un nome e un tipo di dato prima di poter essere usate in un programma. È possibile collocare dovunque dentro `main` ogni definizione di variabile prima del primo uso di quella variabile nel codice. In generale, è opportuno definire le variabili in prossimità del loro primo utilizzo.

Identifieri e maiuscolo/minuscolo

Il nome di una variabile può essere un qualunque **identificatore** valido. Ogni identificatore può essere composto da lettere, cifre e trattini bassi (`_`), ma non può iniziare con una cifra. Il C è **sensibile all'uso del carattere maiuscolo/minuscolo**, quindi `a1` e `A1` sono due differenti identifieri. Un nome di variabile dovrebbe iniziare con una lettera minuscola. Più avanti nel testo, assegneremo un significato speciale agli identifieri che iniziano con una lettera maiuscola e agli identifieri che utilizzano tutte lettere maiuscole.

La scelta di nomi di variabili significativi aiuta a rendere un programma autodocumentante, riducendo quindi il numero di commenti necessari. Evitate di usare come primo carattere di un identificatore un trattino basso (`_`) per evitare conflitti con gli identifieri generati dal compilatore e quelli della Libreria Standard. Anche i nomi di variabili costituiti da più parole possono rendere i programmi più leggibili. Per creare tali nomi:

- separate le parole con trattini bassi, come in `total_commissions`, o
- scrivete le parole attaccate e cominciate ogni nuova parola con una lettera maiuscola, come per esempio `totalCommissions`.

L'ultimo stile – spesso chiamato **notazione a cammello** perché l'alternarsi di lettere maiuscole e minuscole ricorda la silhouette di un cammello – viene preferito.

Messaggi di prompt

La riga 10

```
printf("Enter first integer: "); // prompt
```

stampa "Enter first integer:". Questo messaggio è chiamato **prompt** (letteralmente "richiesta di comando") perché dice all'utente di compiere una specifica azione.

La funzione `scanf` e gli input formattati

La riga 11

```
scanf("%d", &integer1); // legge un intero
```

usa `scanf` per ottenere un valore dall'utente. La funzione legge dallo standard input, che solitamente è la tastiera.

La "f" in `scanf` sta per "formattato". Questa `scanf` ha due argomenti: "%d" e `&integer1`. Il primo, "%d", è la **stringa di controllo del formato**: indica il tipo di dato che deve essere inserito dall'utente. La **specificità**

di conversione %d indica che il dato deve essere un intero (la lettera d sta per “decimale intero”). Un carattere % inizia una specifica di conversione.

Il secondo argomento di scanf inizia con il simbolo & seguito dal nome della variabile. Il simbolo & è l'**operatore di indirizzo** e, se combinato con il nome della variabile, comunica a scanf la locazione (o l'indirizzo) in memoria in cui è contenuta la variabile integer1; scanf quindi memorizza il valore che l'utente inserisce in quella locazione.

L'uso del simbolo & causa spesso confusione ai programmatori inesperti o a coloro che hanno programmato in altri linguaggi che non richiedono questa notazione. Per adesso ricordate soltanto di far precedere ciascuna variabile in ogni chiamata di scanf con il simbolo &. Alcune eccezioni a questa regola saranno esaminate nei Capitoli 6 e 7. L'uso del simbolo & diverrà chiaro dopo che avremo studiato i puntatori nel Capitolo 7.

 Dimenticare di far precedere dal simbolo & una variabile in un'istruzione scanf generalmente provoca un errore al momento dell'esecuzione. Su molti sistemi questo causa un “errore di segmentazione” o una “violatione di accesso”. Un simile errore si verifica quando il programma dell'utente tenta di accedere a una parte della memoria del computer per la quale non ha privilegi d'accesso. La causa precisa di questo errore sarà spiegata nel Capitolo 7.

Quando viene eseguita la riga 11, il computer aspetta che l'utente inserisca un valore per integer1. L'utente digita un intero, poi preme il tasto *Invio* per inviare il numero al computer. Il computer poi inserisce il numero (o valore) in integer1. Qualunque riferimento successivo a integer1 all'interno del programma userà questo stesso valore. Le funzioni printf e scanf facilitano l'interazione tra l'utente e il computer. Questa interazione assomiglia a un dialogo ed è spesso chiamata **elaborazione interattiva**.

Prompt e inserimento del secondo intero

La riga 13

```
printf("Enter second integer: "); // prompt
```

chiede all'utente di inserire il secondo intero, poi la riga 14

```
scanf("%d", &integer2); // legge un intero
```

ottiene dall'utente un valore per la variabile integer2.

Definire la variabile sum

La riga 16

```
int sum = 0; // variabile nella quale viene memorizzata sum
```

definisce la variabile sum di tipo int e la inizializza a 0 prima del suo utilizzo nella riga 17.

Istruzione di assegnazione

L'istruzione di assegnazione nella riga 17

```
sum = integer1 + integer2; // assegna il totale a sum
```

calcola il totale delle variabili integer1 e integer2, quindi assegna il risultato alla variabile sum usando l'**operatore di assegnazione** (=). L'istruzione è letta come “a sum è assegnato il valore dell'espressione integer1 + integer2”. La maggior parte dei calcoli viene eseguita nelle istruzioni di assegnazione.

Operatori binari

L'operatore = e l'operatore + sono chiamati **operatori binari** poiché ciascuno ha due **operandi**. Gli operandi dell'operatore + sono integer1 e integer2. Gli operandi dell'operatore = sono sum e il valore dell'espressione integer1 + integer2. Lasciate spazi su entrambi i lati di un operatore binario per far sì che l'operatore sia in evidenza e rendere il programma più leggibile.

Stampare con una stringa di controllo del formato

La stringa di controllo del formato "Sum is %d\n" nella riga 19

```
printf("Sum is %d\n", sum); // stampa la somma
```

contiene alcuni caratteri letterali da stampare ("Sum is ") e la specifica di conversione %d, che indica che sarà stampato un intero. La somma (sum) è il valore da inserire al posto di %d. La specifica di conversione per un intero (%d) è la stessa sia in printf che in scanf (ciò vale per la maggior parte dei tipi di dati in C).

Combinare la definizione di una variabile e un'istruzione di assegnazione

È possibile assegnare un valore a una variabile nella sua definizione. Per esempio, le righe 16 e 17 possono sommare le variabili integer1 e integer2, poi inizializzare la variabile sum con il risultato:

```
int sum = integer1 + integer2; // assegna il totale a sum
```

Calcoli all'interno delle istruzioni printf

In realtà, non abbiamo bisogno della variabile sum, poiché possiamo eseguire il calcolo all'interno dell'istruzione printf. Quindi, le righe 16-19 possono essere sostituite con

```
printf("Sum is %d\n", integer1 + integer2);
```

✓ Autovalutazione

1. (*Scelta multipla*) Quale tra queste istruzioni chiede correttamente l'input all'utente?

- a) printf("Enter the day of the week: ")
- b) printf(Enter the day of the week:);
- c) printf('Enter the day of the week: ');
- d) printf("Enter the day of the week: ");

Risposta: d.

2. (*Scelta multipla*) L'istruzione seguente si legge come "a sum è assegnato il valore dell'espressione integer1 + integer2". In questa istruzione, = è l'operatore _____.

```
sum = integer1 + integer2;
```

- a) di uguaglianza.
- b) di confronto.
- c) di assegnazione.
- d) Nessuno dei precedenti.

Risposta: c.

2.4 Concetti relativi alla memoria

Ogni variabile ha un nome, un tipo, un valore e una locazione nella memoria del computer. Nel programma di addizione della Figura 2.4, quando la riga 11

```
scanf("%d", &integer1); // legge un intero
```

viene eseguita, il programma inserisce l'input dell'utente nella locazione di memoria di integer1. Supponete che l'utente inserisca 45 come valore di integer1. Concettualmente, la memoria appare come segue:

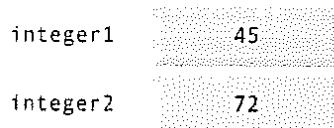


Ogni volta che viene posto un valore in una locazione di memoria, esso sostituisce il precedente in quella locazione, che va perso; per tale motivo, questo processo si dice **distruttivo**.

Tornando al nostro programma per l'addizione, quando la riga 14

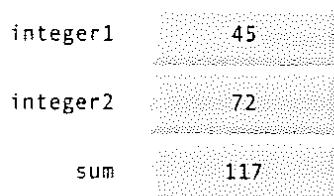
```
scanf("%d", &integer2); // legge un intero
```

viene eseguita, supponete che l'utente inserisca il valore 72. Concettualmente, la memoria appare come segue:



Queste locazioni non sono necessariamente adiacenti nella memoria.

- Una volta che il programma ha ottenuto i valori per `integer1` e `integer2`, la riga 18
- ```
sum = integer1 + integer2; // assegna il totale a sum
```
- somma questi valori e memorizza il totale nella variabile `sum`, sostituendo il suo valore precedente. Concettualmente, la memoria ora appare come segue:



I valori `integer1` e `integer2` rimangono esattamente come erano prima di essere usati nel calcolo: durante il calcolo sono stati usati, ma non distrutti. Pertanto, quando un valore viene semplicemente letto dalla locazione di memoria, il processo si dice **non distruttivo**.

## ✓ Autovalutazione

- 1. (Scelta multipla)** Quale delle seguenti affermazioni a), b) o c) è *falsa*?
  - a) I nomi di variabili corrispondono a locazioni nella memoria del computer.
  - b) Ogni variabile ha un nome, un tipo e un valore.
  - c) Quando un valore è posto in una locazione di memoria, sostituisce il valore precedente in quella locazione. Il valore precedente va perso; per tale motivo, questo processo si dice distruttivo. Quando un valore viene letto da una locazione di memoria, il processo si dice non distruttivo.
  - d) Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

## 2.5 Aritmetica in C

La maggior parte dei programmi in C esegue calcoli usando i seguenti **operatori aritmetici binari**:

| Operazione in C | Operatore aritmetico | Espressione algebrica | Espressione in C   |
|-----------------|----------------------|-----------------------|--------------------|
| Addizione       | +                    | $f + 7$               | <code>f + 7</code> |
| Sottrazione     | -                    | $p - c$               | <code>p - c</code> |
| Moltiplicazione | *                    | $b m$                 | <code>b * m</code> |
| Divisione       | /                    | $x/y$ o $\frac{x}{y}$ | <code>x / y</code> |
| Resto           | %                    | $r \bmod s$           | <code>r % s</code> |

Notate l'uso di vari simboli speciali non usati comunemente in algebra. L'asterisco (\*) indica la moltiplicazione e il segno di percentuale (%) indica l'operatore di resto, che viene introdotto di seguito. In algebra, per moltiplicare  $a$  per  $b$ , mettiamo semplicemente l'uno accanto all'altro questi nomi di variabile costituiti da singole lettere, come in  $ab$ . In C,  $ab$  sarebbe interpretato come un nome singolo di due lettere (o identificatore). La maggior parte dei linguaggi di programmazione richiede che la moltiplicazione sia indicata esplicitamente usando l'operatore \*, come in  $a * b$ .

### Divisione intera e operatore di resto

La divisione intera (ovvero, dividere un intero per un altro) restituisce un risultato intero, quindi  $7 / 4$  restituisce 1, e  $17 / 5$  restituisce 3. L'operatore di resto % calcola il resto di una divisione intera, quindi  $7 \% 4$  restituisce 3 e  $17 \% 5$  restituisce 2. Esamineremo in seguito molte interessanti applicazioni dell'operatore di resto.

 Un tentativo di dividere per zero porta a un risultato indefinito sui computer e generalmente provoca un errore irreversibile, ossia un errore che causa l'interruzione immediata di un programma, senza che questo porti a termine con successo il suo lavoro. Gli errori non irreversibili permettono ai programmi di completare l'esecuzione, producendo spesso risultati scorretti.

### Espressioni aritmetiche in forma lineare

Le espressioni aritmetiche devono essere scritte in **forma lineare** per facilitare l'inserimento di programmi nel computer. Espressioni come "a diviso b" devono essere scritte come  $a/b$  in modo che tutti gli operatori e gli operandi compaiano in una configurazione lineare. La notazione algebrica

$$\frac{a}{b}$$

non è generalmente accettabile per i compilatori, sebbene alcuni pacchetti software specifici tollerino una notazione più naturale per le espressioni matematiche complesse.

### Parentesi per raggruppare sottoespressioni

Le parentesi sono usate nelle espressioni in C allo stesso modo che nelle espressioni algebriche. Per esempio, per moltiplicare per  $a$  la quantità  $b + c$ , scriviamo  $a * (b + c)$ .

### Regole di precedenza degli operatori

Il C applica gli operatori nelle espressioni aritmetiche in una precisa sequenza, determinata dalle seguenti **regole di precedenza degli operatori**, che generalmente sono le stesse di quelle dell'algebra:

1. Le espressioni raggruppate tra parentesi vengono valutate per prime. Le parentesi si considerano al "massimo livello di precedenza". In caso di **parentesi annidate**, come

$$((a + b) + c)$$

gli operatori nella coppia di parentesi più interne sono applicati per primi.

2. Le operazioni con \*, / e % sono calcolate subito dopo. Se un'espressione contiene diversi operatori \*, / e %, il calcolo procede da sinistra a destra. Questi tre operatori hanno lo stesso livello di precedenza.
3. Segue il calcolo delle operazioni con + e -. Se un'espressione contiene operatori + e -, il calcolo procede da sinistra a destra. Questi due operatori hanno lo stesso livello di precedenza, che è inferiore di quello di \*, / e %.
4. L'operatore di assegnazione (=) è valutato per ultimo.

Le regole di precedenza degli operatori specificano l'ordine che viene usato per calcolare le espressioni in C.<sup>1</sup> Quando diciamo che il calcolo procede da sinistra a destra, ci riferiamo all'**associatività** degli operatori. Alcuni operatori sono associativi da destra a sinistra.

1. Usiamo esempi semplici per spiegare l'ordine di calcolo delle espressioni. Problemi particolari si presentano in espressioni più complesse che incontrerete in seguito nel libro. Affronteremo questi problemi quando compariranno.

**Esempi di espressioni algebriche e di espressioni in C**

Adesso consideriamo diverse espressioni alla luce delle regole di precedenza degli operatori. Ogni esempio presenta un'espressione algebrica e la sua equivalente in C. La seguente espressione calcola la media aritmetica di cinque termini.

Algebra:  $m = \frac{a + b + c + d + e}{5}$

C:  $m = (a + b + c + d + e) / 5;$

Nell'istruzione C sono richieste le parentesi per raggruppare le addizioni perché la divisione ha una precedenza più alta dell'addizione. L'intera quantità  $(a + b + c + d + e)$  deve essere divisa per 5. Se le parentesi sono erroneamente omesse, otteniamo  $a + b + c + d + e / 5$ , che viene calcolata non correttamente come

$$a + b + c + d + \frac{e}{5}$$

La seguente espressione è l'equazione di una linea retta:

Algebra:  $y = mx + b$

C:  $y = m * x + b;$

Non sono richieste parentesi. La moltiplicazione è calcolata per prima perché ha una precedenza più alta dell'addizione.

La seguente espressione contiene le operazioni di resto (%), moltiplicazione, divisione, addizione, sottrazione e assegnazione:

Algebra:  $z = pr \bmod q + w/x - y$

C:  $z = p * r \% q + w / x - y;$



I numeri cerchiati indicano l'ordine in cui vengono applicati gli operatori in C. La moltiplicazione, il resto e la divisione sono calcolati per primi nell'ordine da sinistra a destra (cioè sono associativi da sinistra a destra) perché hanno una precedenza più alta dell'addizione e della sottrazione. L'addizione e la sottrazione sono calcolate successivamente, anch'esse da sinistra a destra. Alla fine il risultato è assegnato alla variabile z.

**Parentesi "allo stesso livello"**

Non tutte le espressioni con diverse coppie di parentesi contengono parentesi annidate. Nell'espressione seguente, le parentesi sono "allo stesso livello":

$$a * (b + c) + c * (d + e)$$

In questo caso, le espressioni tra parentesi vengono valutate da sinistra a destra.

**Calcolo di un polinomio di secondo grado**

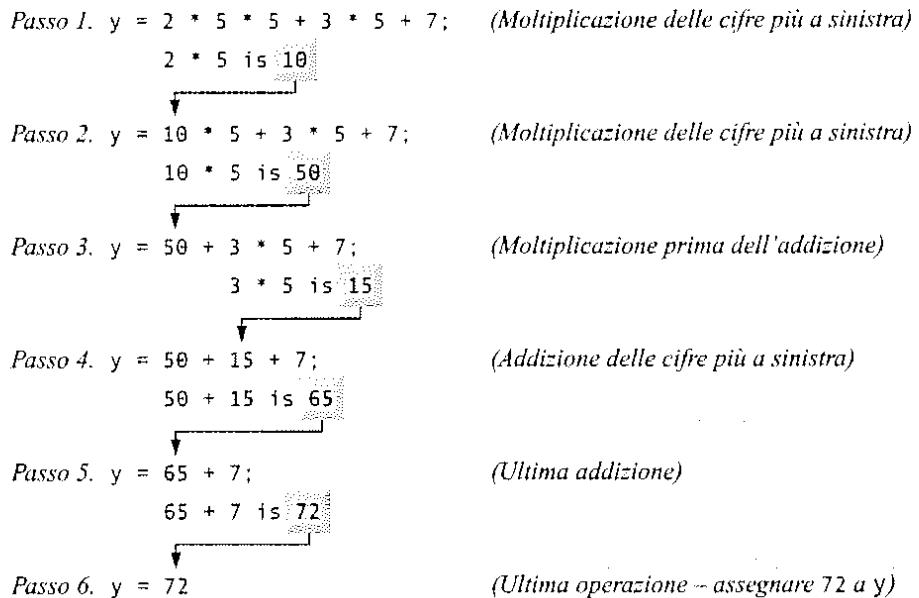
Per comprendere meglio le regole di precedenza degli operatori, vediamo come viene calcolato un polinomio di secondo grado in C.

$y = a * x * x + b * x + c;$



I numeri cerchiati sotto l'istruzione indicano l'ordine in cui vengono eseguite le operazioni in C. Non c'è alcun operatore aritmetico per l'esponente in C, così abbiamo scritto  $x^2$  come  $x * x$ . La funzione pow ("power") della Libreria Standard consente di calcolare una potenza, come vedremo nel Capitolo 4.

Supponete che nel precedente polinomio di secondo grado  $a = 2$ ,  $b = 3$ ,  $c = 7$  e  $x = 5$ . Il seguente diagramma illustra l'ordine in cui gli operatori sono applicati:



### Utilizzare parentesi per chiarezza

Come in algebra, è accettabile usare **parentesi ridondanti** per rendere più chiara un'espressione. Quindi, l'istruzione precedente potrebbe essere scritta con parentesi come segue:

$$y = (a * x * x) + (b * x) + c;$$

### ✓ Autovalutazione

1. *(Scelta multipla)* C'è qualcuna tra le seguenti espressioni che esegue correttamente il calcolo in C "somma 3 alla quantità 4 per 5"?
  - $3 + 4 * 5$
  - $3 + (4 * 5)$
  - $(3 + (4 * 5))$
  - Tutte le precedenti.

Risposta: d.

2. *(Scelta multipla)* Considerate l'istruzione:

$$y = a * x * x + b * x + c;$$

Quali delle seguenti variazioni dell'istruzione precedente contengono parentesi ridondanti?

- $y = (a * x * x + b * x + c);$
- $y = a * (x * x) + (b * x) + c;$
- $y = (a * x * x) + (b * x) + c;$
- Tutte le precedenti.

Risposta: d.

## 2.6 Decisioni: operatori di uguaglianza e relazionali

Le istruzioni eseguibili o effettuano azioni come calcoli, input e output, oppure, come vedremo, prendono **decisioni**. Per esempio, un programma potrebbe determinare se il voto di una persona a un esame sia maggiore o uguale a 60, e di conseguenza decidere se stampare il messaggio “Congratulazioni! Sei stato promosso”.

Una **condizione** è un'espressione che può essere *vera* (cioè la condizione è soddisfatta,) o *falsa* (cioè la condizione non è soddisfatta). Questa sezione introduce l'**istruzione if**, che permette a un programma di prendere una decisione basata sul valore di una condizione. Se la condizione è *vera*, l'istruzione nel corpo dell'istruzione **if** viene eseguita, altrimenti non viene eseguita.

### Operatori di uguaglianza e relazionali

Le condizioni vengono costruite usando i seguenti **operatori di uguaglianza e relazionali**:

| Operatore di uguaglianza o relazionale in algebra | Operatore di uguaglianza o relazionale in C | Esempio di condizione in C | Significato della condizione in C |
|---------------------------------------------------|---------------------------------------------|----------------------------|-----------------------------------|
| <i>Operatori relazionali</i>                      |                                             |                            |                                   |
| >                                                 | >                                           | x > y                      | x è maggiore di y                 |
| <                                                 | <                                           | x < y                      | x è minore di y                   |
| $\geq$                                            | $\geq$                                      | x $\geq$ y                 | x è maggiore o uguale a y         |
| $\leq$                                            | $\leq$                                      | x $\leq$ y                 | x è minore o uguale a y           |
| <i>Operatori di uguaglianza</i>                   |                                             |                            |                                   |
| =                                                 | $\equiv$                                    | x == y                     | x è uguale a y                    |
| $\neq$                                            | $\neq$                                      | x != y                     | x non è uguale a y                |

Gli operatori relazionali  $<$ ,  $\leq$ ,  $>$  e  $\geq$  hanno tutti lo stesso livello di precedenza e sono associativi da sinistra a destra. Gli operatori di uguaglianza  $\equiv$  e  $\neq$  hanno lo stesso livello di precedenza, che è inferiore di quello degli operatori relazionali, e anch'essi sono associativi da sinistra a destra. In C, una condizione può essere effettivamente qualsiasi espressione che genera un valore zero (*falso*) o un valore diverso da zero (*vero*).

### Confondere l'operatore di uguaglianza == con l'operatore di assegnazione

Confondere == con l'operatore di assegnazione (=) è un comune errore di programmazione. Per evitare questa confusione, l'operatore di uguaglianza va letto “doppio uguale” e l'operatore di assegnazione va letto “assume il valore”. Come vedrete, confondere questi operatori può causare errori logici molto insidiosi piuttosto che errori di compilazione.

### Dimostrare l'istruzione if

La Figura 2.5 usa sei istruzioni if per confrontare due numeri inseriti dall'utente. Per ciascuna istruzione if con una condizione *vera*, viene eseguita la corrispondente istruzione printf. Il programma e i risultati in uscita dei tre esempi di esecuzione sono mostrati nella figura.

```

1 // fig02_05.c
2 // Uso di istruzioni if, operatori relazionali
3 // e operatori di uguaglianza.
4 #include <stdio.h>
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8 printf("Enter two integers, and I will tell you\n");

```

```

9 printf("the relationships they satisfy: ");
10
11 int number1 = 0; // primo numero inserito dall'utente
12 int number2 = 0; // secondo numero inserito dall'utente
13
14 scanf("%d %d", &number1, &number2); // legge due interi
15
16 if (number1 == number2) {
17 printf("%d is equal to %d\n", number1, number2);
18 } // fine di if
19
20 if (number1 != number2) {
21 printf("%d is not equal to %d\n", number1, number2);
22 } // fine di if
23
24 if (number1 < number2) {
25 printf("%d is less than %d\n", number1, number2);
26 } // fine di if
27
28 if (number1 > number2) {
29 printf("%d is greater than %d\n", number1, number2);
30 } // fine di if
31
32 if (number1 <= number2) {
33 printf("%d is less than or equal to %d\n", number1, number2);
34 } // fine di if
35
36 if (number1 >= number2) {
37 printf("%d is greater than or equal to %d\n", number1, number2);
38 } // fine di if
39 } // fine della funzione main

```

Enter two integers, and I will tell you  
 the relationships they satisfy: 3 7  
 3 is not equal to 7  
 3 is less than 7  
 3 is less than or equal to 7

Enter two integers, and I will tell you  
 the relationships they satisfy: 22 12  
 22 is not equal to 12  
 22 is greater than 12  
 22 is greater than or equal to 12

Enter two integers, and I will tell you  
 the relationships they satisfy: 7 7  
 7 is equal to 7  
 7 is less than or equal to 7  
 7 is greater than or equal to 7

**Figura 2.5** Uso di istruzioni if, operatori relazionali e operatori di uguaglianza.

Il programma usa `scanf` (riga 14) per leggere due interi nelle variabili `int number1` e `number2`. Il primo `%d` converte un valore da memorizzare nella variabile `number1`. Il secondo converte un valore da memorizzare nella variabile `number2`.

### Confronto di numeri

L'istruzione `if` nelle righe 16-18

```
if (number1 == number2) {
 printf("%d is equal to %d\n", number1, number2);
} // fine di if
```

confronta i valori delle variabili `number1` e `number2` per vedere se sono uguali. Se i valori sono uguali, la riga 17 stampa una riga di testo che afferma che i numeri sono uguali. Per ogni condizione *vera* nelle istruzioni `if` nelle righe 20, 24, 28, 32 e 36, la corrispondente istruzione del corpo stampa una riga di testo. Indentare il corpo di ognuna delle istruzioni `if` e inserire righe vuote sopra e sotto di esse migliora la leggibilità del programma.

Una parentesi graffa sinistra, `{`, inizia il corpo di ognuna delle istruzioni `if` (es. la riga 16). Una parentesi graffa destra corrispondente, `}`, termina il corpo di ognuna delle istruzioni `if` (es. la riga 18). Nel corpo di un'istruzione `if` si può inserire un numero qualsiasi di istruzioni.<sup>2</sup>

Inserire un punto e virgola immediatamente dopo la parentesi destra che segue la condizione di un'istruzione `if` è un errore comune. In questo caso, il punto e virgola viene trattato come un'istruzione vuota che non esegue alcuna attività – l'istruzione che doveva essere parte del corpo dell'istruzione `if` non lo è più e viene sempre eseguita.

### Operatori introdotti fin qui

La tabella seguente elenca la precedenza degli operatori introdotti fin qui, dalla più alta alla più bassa.

| Operatori                                                                 | Associatività        |
|---------------------------------------------------------------------------|----------------------|
| <code>()</code>                                                           | da sinistra a destra |
| <code>*</code> <code>/</code> <code>%</code>                              | da sinistra a destra |
| <code>+</code> <code>-</code>                                             | da sinistra a destra |
| <code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>&gt;=</code> | da sinistra a destra |
| <code>==</code> <code>!=</code>                                           | da sinistra a destra |
| <code>=</code>                                                            | da destra a sinistra |

L'operatore di assegnazione (`=`) associa da destra a sinistra. Quando scrivete espressioni contenenti molti operatori, fate riferimento alla tabella di precedenza degli operatori. Controllate che gli operatori nell'espressione siano applicati nell'ordine giusto. Se non siete sicuri sull'ordine di calcolo in un'espressione complessa, usate le parentesi per raggruppare le espressioni o spezzate l'istruzione in diverse istruzioni più semplici.

### Parole chiave

Alcune parole che abbiamo usato negli esempi di questo capitolo, come `int`, `if` e `void`, sono **parole chiave** o parole riservate del linguaggio e hanno un significato speciale per il compilatore. La tabella seguente contiene le parole chiave del C. Non usatele come identificatori.

2. L'uso di parentesi graffe per delimitare il corpo di un'istruzione `if` è facoltativo quando il corpo contiene soltanto un'istruzione, ma è considerata una buona pratica usarle sempre. Nel Capitolo 3 parleremo di questo argomento.

| <b>Parole chiave</b>                                                       |        |          |         |          |
|----------------------------------------------------------------------------|--------|----------|---------|----------|
| auto                                                                       | do     | goto     | signed  | unsigned |
| break                                                                      | double | if       | sizeof  | void     |
| case                                                                       | else   | int      | static  | volatile |
| char                                                                       | enum   | long     | struct  | while    |
| const                                                                      | extern | register | switch  |          |
| continue                                                                   | float  | return   | typedef |          |
| default                                                                    | for    | short    | union   |          |
| <i>Parole chiave aggiunte nello standard C99</i>                           |        |          |         |          |
| _Bool _Complex _Imaginary inline restrict                                  |        |          |         |          |
| <i>Parole chiave aggiunte nello standard C11</i>                           |        |          |         |          |
| _Alignas _Alignnof _Atomic _Generic _Noreturn _Static_assert _Thread_local |        |          |         |          |

## ✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) Una condizione è un'espressione che può essere *vera* o *falsa*.
- b) Una condizione può essere qualsiasi espressione che genera un valore zero (*vero*) o un valore diverso da zero (*falso*).
- c) L'istruzione `if` prende una decisione basata sul valore di una condizione. Se la condizione è *vera*, l'istruzione nel corpo dell'istruzione `if` viene eseguita; altrimenti, non viene eseguita.
- d) Le condizioni vengono costruite usando gli operatori di uguaglianza e gli operatori relazionali.

**Risposta:** b) è *falsa*. In realtà, in C, una condizione può essere qualsiasi espressione che genera un valore zero (*falso*) o un valore diverso da zero (*vero*).

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) Il corpo della seguente istruzione `if` viene eseguito se `number1` è uguale a `number2`:

```
if (number1 == number2) {
 printf("%d is equal to %d\n", number1, number2);
} // fine di if
```

- b) Se non siete sicuri sull'ordine di calcolo in un'espressione complessa, usate le parentesi per raggruppare le espressioni o spezzate l'istruzione in diverse istruzioni più semplici.
- c) Nel corpo di un'istruzione `if` si può inserire un numero qualsiasi di istruzioni. L'uso di parentesi graffe per delimitare il corpo di un'istruzione `if` è obbligatorio.
- d) Alcuni operatori del C, come l'operatore di assegnazione (`=`), sono associativi da destra a sinistra piuttosto che da sinistra a destra.

**Risposta:** c) è *falsa*. In realtà, l'uso di parentesi graffe per delimitare il corpo di un'istruzione `if` è facoltativo quando il corpo contiene soltanto un'istruzione. In ogni caso, usare sempre queste parentesi aiuta a evitare errori.



## 2.7 Programmazione sicura in C

Nella Prefazione abbiamo fatto menzione del *SEI CERT C Coding Standard* e abbiamo detto che avremmo seguito certe linee guida che vi avrebbero aiutato a evitare pratiche di programmazione che espongono i sistemi ad attacchi.

### Evitare printf con argomento singolo

Una linea guida consiste nell'evitare l'uso di `printf` con una sola stringa come argomento.<sup>3</sup> Il primo argomento di `printf` è una stringa di formattazione, che `printf` analizza per le specifiche di conversione. Sostituisce poi ogni specifica di conversione con il valore di un argomento successivo; prova a farlo anche se non c'è un argomento successivo che possa essere usato.

In uno dei prossimi capitoli, imparerete come ricevere in ingresso stringhe dagli utenti. Sebbene il primo argomento di `printf` sia solitamente un stringa letterale, potrebbe anche essere una variabile contenente una stringa inserita da un utente. In tali casi, un malintenzionato potrebbe realizzare una stringa di formattazione come input da utente che abbia più specifiche di conversione rispetto agli argomenti aggiuntivi di `printf`. Questo exploit è stato usato dagli autori di attacchi per leggere memoria a cui non dovrebbero poter accedere.<sup>4</sup>

Ci sono un paio di misure preventive che consentono di evitare attacchi di questo tipo. Se dovete stampare una stringa che termina con un newline, usate al posto di `printf` la **funzione `puts`**, che stampa il suo argomento stringa seguito da un newline. Per esempio, nella Figura 2.1, la riga 7

```
printf("Welcome to C!\n");
```

andrebbe scritta come

```
puts("Welcome to C!");
```

La funzione `puts` stampa semplicemente il contenuto del suo argomento stringa, quindi una specifica di conversione verrebbe stampata come i suoi singoli caratteri.

Per stampare una stringa senza un carattere newline di terminazione, usate `printf` con due argomenti: una stringa di controllo per il formato "%s" e la stringa da stampare. La **specificia di conversione %s** è un segnaposto per una stringa. Per esempio, nella Figura 2.2, la riga 7

```
printf("Welcome ");
```

andrebbe scritta come

```
printf("%s", "Welcome ");
```

Come avviene con `puts`, se il secondo argomento di `printf` contiene una specifica di conversione, verrà stampato come i suoi singoli caratteri.

Sebbene le `printf` in questo capitolo così come sono scritte siano in realtà sicure, tali modifiche sono buone pratiche di programmazione che eliminaranno certe vulnerabilità relative alla sicurezza quando andremo più a fondo nel linguaggio C (nel seguito del libro spiegheremo il motivo). D'ora in poi useremo queste pratiche nei nostri esempi e voi dovrete usarle nel vostro codice.

### `scanf`, `printf`, `scanf_s` e `printf_s`

Diremo di più sulle funzioni `scanf` e `printf` nei successivi paragrafi "Programmazione sicura in C", a partire dal Paragrafo 3.13. Esamineremo anche le funzioni `scanf_s` e `printf_s`, che sono state introdotte nel C11 per cercare di eliminare diverse vulnerabilità relative alla sicurezza di `scanf` e `printf`. In un successivo paragrafo "Programmazione sicura in C", discuteremo le ben note vulnerabilità di sicurezza della funzione `scanf` e come evitarle.

## Autovalutazione

1. (*Codice*) Riscrivete l'istruzione seguente come istruzione sicura equivalente usando la funzione `puts`:

```
printf("Enter your age:\n");
```

3. Per ulteriori informazioni, vedi la regola CERT FIO30-C (<https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings>). Nel paragrafo "Programmazione sicura in C" del Capitolo 6 spiegheremo la nozione di input dell'utente come è definita da questa linea guida CERT.

4. "Format String Attack", Format String Software Attack | OWASP Foundation. Accesso luglio 2020. [https://owasp.org/www-community/attacks/Format\\_string\\_attack](https://owasp.org/www-community/attacks/Format_string_attack).

**Risposta:** `puts("Enter your age:");`

2. (*Codice*) Riscrivete l'istruzione seguente come istruzione sicura equivalente usando la funzione `printf`:

```
printf("Enter your age:");
```

**Risposta:** `printf("%s", "Enter your age");`

## 2.8 Riepilogo

In questo capitolo abbiamo introdotto diverse importanti caratteristiche della programmazione in linguaggio C, come la stampa di dati sullo schermo, l'inserimento di dati da parte dell'utente, l'esecuzione di calcoli e il prendere decisioni. Nel prossimo capitolo utilizzeremo queste tecniche nel loro insieme quando introdurremo la programmazione strutturata. Le tecniche di indentazione vi diventeranno più familiari. Studieremo come specificare l'*ordine di esecuzione delle istruzioni*, ovvero il cosiddetto **flusso di controllo**.

### Paragrafo 2.1 Introduzione

- Il linguaggio C facilita un approccio strutturato e disciplinato alla progettazione di programmi per computer.

### Paragrafo 2.2 Un semplice programma in C: stampare una riga di testo

- I commenti cominciano con `//`. Essi documentano i programmi e ne migliorano la leggibilità. I commenti mult linea cominciano con `/*` e finiscono con `*/`.
- I commenti vengono ignorati dal compilatore.
- Il preprocessore elabora le righe che iniziano con `#` prima che il programma sia compilato. La direttiva `#include` dice al preprocessore di includere il contenuto di un altro file.
- Il file di intestazione `<stdio.h>` contiene informazioni usate dal compilatore per assicurare un corretto utilizzo delle funzioni di input/output della Libreria Standard, come `printf`.
- La funzione `main` fa parte di ogni programma. Le parentesi dopo `main` indicano che `main` è un blocco costituente di un programma chiamato **funzione**. I programmi contengono una o più funzioni, una delle quali deve essere `main`, che è quella da cui inizia l'esecuzione del programma.
- Le funzioni possono restituire informazioni. La parola chiave `int` alla sinistra di `main` indica che `main` "restituisce" un valore intero (numero intero).
- Le funzioni possono ricevere informazioni quando vengono invocate. Il `void` tra parentesi dopo `main` indica che `main` non riceve informazioni.
- Una parentesi graffa sinistra, `{`, inizia il **corpo** di ogni funzione. Una corrispondente parentesi graffa destra, `}`, termina ciascuna funzione. La coppia di parentesi graffe e il codice contenuto al loro interno costituiscono un **blocco**.
- La funzione `printf` istruisce il computer a stampare informazioni sullo schermo.
- Una stringa è talvolta chiamata **stringa di caratteri, messaggio o letterale**.
- Ogni istruzione deve terminare con un punto e virgola, noto come **terminatore dell'istruzione**.
- In `\n`, il backslash (`\`) è un **carattere di escape**. Quando si incontra un backslash in una stringa, il compilatore lo combina con il carattere successivo per formare una **sequenza di escape**. La sequenza di escape `\n` significa **newline**, ovvero ritorno a capo in una riga successiva.
- Quando un newline appare nella stringa inviata in uscita da una `printf`, il cursore viene posizionato all'inizio della riga successiva.
- Il **doppio backslash (\ \ ) della sequenza di escape** può essere usato per inserire un singolo backslash in una stringa.
- La sequenza di escape `\ "` rappresenta il carattere "doppiie virgolette".

### Paragrafo 2.3 Un altro semplice programma in C: sommare due interi

- Una **variabile** è una locazione di memoria dove un valore può essere memorizzato per essere usato da un programma.
- Variabili di tipo **int** contengono **valori interi**, cioè **numeri interi**.
- Tutte le variabili devono essere definite con un nome e un **tipo** prima di poter essere utilizzate in un programma.
- Il nome di una variabile in C è qualsiasi **identificatore** valido. Un identificatore è una serie di caratteri costituita da lettere, cifre e trattini bassi (`_`) che non inizia con una cifra.
- Il C fa **distinzione tra maiuscole e minuscole**.
- La **funzione scanf** può essere usata per ricevere dati in ingresso dallo standard input, che solitamente è la tastiera.
- La **stringa di controllo del formato di scanf** indica i tipi di dati da inserire.
- La **specifica di conversione %d** indica un intero (la lettera d sta per “intero decimale”). Il simbolo % inizia una specifica di conversione.
- Gli argomenti che seguono la stringa di controllo per il formato di `scanf` iniziano con il simbolo & seguito dal nome della variabile. In questo contesto, il simbolo & – chiamato **operatore di indirizzo** – indica a `scanf` la locazione di memoria associata alla variabile. Il computer poi memorizza il valore in tale locazione.
- La maggior parte dei calcoli è eseguita nelle **istruzioni di assegnazione**.
- L'operatore = e l'operatore + sono **operatori binari**, ciascuno con due operandi.
- In una `printf` che specifica come suo primo argomento una stringa di controllo per il formato, le specifiche di conversione indicano i segnaposto per i dati da inviare in uscita.

### Paragrafo 2.4 Concetti relativi alla memoria

- Ogni **variabile** ha un **nome**, un **tipo**, un **valore** e una locazione di memoria.
- Tutte le volte che un valore è posto in una locazione di memoria, sostituisce il valore precedente, che viene perso. Pertanto questa operazione è detta **distruttiva**.
- La lettura di un valore da una locazione di memoria è un'operazione **non distruttiva**.

### Paragrafo 2.5 Aritmetica in C

- La maggior parte dei linguaggi di programmazione indicano la moltiplicazione con l'operatore \*, come in `a * b`.
- Le **espressioni aritmetiche** devono essere scritte in **forma lineare** per facilitare l'inserimento di programmi nel computer.
- Le **parentesi** sono usate per raggruppare termini nelle espressioni in C allo stesso modo che nelle espressioni algebriche.
- Il C calcola le espressioni aritmetiche in una sequenza precisa determinata dalle seguenti **regole di precedenza degli operatori**, che generalmente equivalgono a quelle seguite nell'algebra.
- Le espressioni contenenti più operazioni con +, / e % vengono calcolate da sinistra a destra. Questi tre operatori sono allo stesso livello di precedenza.
- Le espressioni contenenti più operazioni con + e - vengono calcolate da sinistra a destra. Questi due operatori hanno lo stesso livello di precedenza, che è inferiore di quello di \*, / e %.
- L'**associatività** degli operatori specifica se essi sono valutati da sinistra a destra o da destra a sinistra.

### Paragrafo 2.6 Decisioni: operatori di uguaglianza e relazionali

- Le istruzioni eseguibili del C o eseguono **azioni** o prendono **decisioni**.
- L'**istruzione if** del C permette a un programma di prendere una decisione basandosi sul fatto che una condizione sia *vera* o *falsa*. Se la condizione è *vera*, il corpo dell'istruzione **if** viene eseguito; altrimenti, non viene eseguito.
- Le condizioni nelle istruzioni **if** vengono scritte usando gli **operatori di uguaglianza e relazionali**.
- Gli operatori relazionali hanno tutti lo stesso livello di precedenza e sono associativi da sinistra a destra. Gli operatori di uguaglianza hanno un livello di precedenza inferiore degli operatori relazionali e sono anch'essi associativi da sinistra a destra.
- Per evitare di confondere l'assegnazione (=) e l'uguaglianza (==), l'operatore di assegnazione andrebbe letto "assume il valore" e l'operatore di uguaglianza "doppio uguale."
- Il compilatore solitamente ignora i **caratteri di spaziatura** come tabulazioni, newline e spazi.
- Le **parole chiave** (o parole riservate) hanno un significato speciale per il compilatore C, per cui non potete utilizzarle come identificatori, come nel caso dei nomi di variabili.

### Paragrafo 2.7 Programmazione sicura in C

- Una buona pratica che contribuisce a evitare che i sistemi rimangano esposti agli attacchi è quella di non usare **printf** con un solo argomento costituito da una singola stringa.
- Per stampare una stringa seguita da un carattere newline, usate la **funzione puts**, che stampa il suo argomento (una stringa) seguito da un carattere newline.
- Per stampare una sola stringa senza il carattere newline finale, potete usare **printf** con la specifica di conversione "%s" come primo argomento e la stringa da stampare come secondo argomento.

### Esercizi di autovalutazione

- 2.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
  - a) Ogni programma in C inizia l'esecuzione dalla funzione \_\_\_\_\_.
  - b) Il corpo di ogni funzione inizia con \_\_\_\_\_ e termina con \_\_\_\_\_.
  - c) Ogni istruzione termina con un \_\_\_\_\_.
  - d) La funzione della Libreria Standard \_\_\_\_\_ stampa le informazioni sullo schermo.
  - e) La sequenza di escape \n rappresenta il carattere di \_\_\_\_\_, che porta il cursore a posizionarsi all'inizio della riga successiva sullo schermo.
  - f) La funzione della Libreria Standard \_\_\_\_\_ è usata per leggere dati da tastiera.
  - g) La specifica di conversione \_\_\_\_\_ in una stringa di controllo per il formato in un'istruzione **printf** o **scanf** indica rispettivamente che verrà stampato o letto un intero.
  - h) Ogni volta che un nuovo valore è posto in una locazione di memoria, esso annulla il valore precedente in quella locazione. Questa operazione si dice \_\_\_\_\_.
  - i) Quando un valore è letto da una locazione di memoria, il valore in quella locazione viene lasciato inalterato; questa operazione si dice \_\_\_\_\_.
  - j) L'istruzione \_\_\_\_\_ è usata per prendere decisioni.
- 2.2 Stabilite se ognuna delle seguenti frasi è *vera* o *falsa*. Se *falsa*, spiegate perché.
  - a) La funzione **printf** comincia a stampare sempre all'inizio di una nuova riga.
  - b) I commenti fanno sì che il computer stampi sullo schermo il testo dopo // durante l'esecuzione del programma.
  - c) La sequenza di escape \n, quando è usata in una stringa di controllo per il formato in un'istruzione **printf**, fa posizionare il cursore di output all'inizio della riga successiva sullo schermo.
  - d) Tutte le variabili devono essere definite prima del loro utilizzo.
  - e) A tutte le variabili deve essere assegnato un tipo quando sono definite.
  - f) Il linguaggio C considera identiche le variabili **number** e **NuMbEr**.

- g) Le definizioni possono comparire ovunque nel corpo di una funzione.  
 h) Tutti gli argomenti che seguono la stringa di controllo per il formato in una funzione `printf` devono essere preceduti dal simbolo &.  
 i) L'operatore di resto (%) può essere usato soltanto con operandi interi.  
 j) Gli operatori aritmetici \*, /, %, + e - hanno tutti lo stesso livello di precedenza.  
 k) Un programma che stampa tre righe di output deve contenere tre istruzioni `printf`.
- 2.3 Scrivete singole istruzioni in C per realizzare ognuna delle seguenti operazioni.
- Definite la variabile `number` di tipo `int` e inizializzatela a 0.
  - Chiedete all'utente di inserire un intero. Terminate il vostro messaggio di richiesta con i due punti (:) seguiti da uno spazio e lasciate il cursore posizionato dopo lo spazio.
  - Leggete un intero da tastiera e memorizzate il valore in una variabile intera `a`.
  - Se `number` non è uguale a 7, stampate "number is not equal to 7".
  - Stampate "This is a C program." su una riga.
  - Stampate "This is a C program." su due righe in modo che la prima riga termini con C.
  - Stampate "This is a C program." con ogni parola su una riga separata.
  - Stampate "This is a C program." con le parole separate da tabulazioni.
- 2.4 Scrivete un'istruzione (o un commento) per realizzare ognuna delle seguenti operazioni.
- Dichiarate che un programma calcolerà il prodotto di tre interi.
  - Chiedete all'utente di inserire tre interi.
  - Definite la variabile `x` di tipo `int` e inizializzatela a 0.
  - Definite la variabile `y` di tipo `int` e inizializzatela a 0.
  - Definite la variabile `z` di tipo `int` e inizializzatela a 0.
  - Leggete tre interi da tastiera e memorizzateli nelle variabili `x`, `y` e `z`.
  - Definite la variabile `result`, calcolate il prodotto degli interi nelle variabili `x`, `y` e `z`, quindi utilizzate il risultato per inizializzare la variabile `result`.
  - Stampate "The product is" seguito dal valore della variabile intera `result`.
- 2.5 Usando le istruzioni che avete scritto nell'Esercizio 2.4, scrivete un programma completo che calcoli il prodotto di tre interi.
- 2.6 Identificate e correggete gli errori in ciascuna delle seguenti istruzioni.
- `printf("The value is %d\n", &number);`
  - `scanf("%d%d", &number1, number2);`
  - `if (c < 7);{`  
      `puts("C is less than 7");`  
}
  - `if (c => 7) {`  
      `puts("C is greater than or equal to 7");`  
}
- Risposte agli esercizi di autovalutazione**
- 2.1 a) main. b) parentesi graffa sinistra ({), parentesi graffa destra (}). c) punto e virgola. d) `printf`. e) newline. f) `scanf`. g) `%d`. h) distruttiva. i) non distruttiva. j) `if`.
- 2.2 a) Falso. La funzione `printf` comincia a stampare sempre da dove il cursore è posizionato, e questa posizione può essere ovunque su una riga dello schermo.  
 b) Falso. I commenti non fanno effettuare alcuna azione durante l'esecuzione del programma; sono usati per documentare i programmi e migliorarne la leggibilità.  
 c) Vero.  
 d) Vero.  
 e) Vero.

- f) *Falso.* Il linguaggio C fa distinzione tra maiuscole e minuscole, pertanto queste variabili sono diverse.  
 g) *Vero.*  
 h) *Falso.* Gli argomenti in una funzione `printf` generalmente non devono essere preceduti dal simbolo `&`. Gli argomenti che seguono la stringa di controllo del formato in una funzione `scanf` generalmente devono essere preceduti da un `&`. Analizzeremo le eccezioni a queste regole nei Capitoli 6 e 7.  
 i) *Vero.*  
 j) *Falso.* Gli operatori `*`, `/` e `%` sono allo stesso livello di precedenza e gli operatori `+` e `-` sono a un livello di precedenza inferiore.  
 k) *Falso.* Un'istruzione `printf` con più sequenze di escape `\n` può stampare diverse righe.
- 2.3 a) `int number = 0;`  
 b) `printf("%s", "Enter an integer: ");`  
 c) `scanf("%d", &a);`  
 d) `if (number != 7) {`  
     `puts("The variable number is not equal to 7.");`  
     `}`  
 e) `puts("This is a C program.");`  
 f) `puts("This is a C\nprogram.");`  
 g) `puts("This\nis\na\nC\nprogram.");`  
 h) `puts("This\tis\tta\ttC\tprogram.");`
- 2.4 a) // Calcola il prodotto di tre interi  
 b) `printf("%s", "Enter three integers: ");`  
 c) `int x;`  
 d) `int y;`  
 e) `int z;`  
 f) `scanf("%d%d%d", &x, &y, &z);`  
 g) `int result = x * y * z;`  
 h) `printf("The product is %d\n", result);`
- 2.5 Si veda sotto.
- ```

1 // Calcola il prodotto di tre interi
2 #include <stdio.h>
3
4 int main(void) {
5     printf("Enter three integers: "); // prompt
6
7     int x = 0;
8     int y = 0;
9     int z = 0;
10    scanf("%d%d%d", &x, &y, &z); // legge tre interi
11
12    int result = x * y * z; // moltiplica i valori
13    printf("The product is %d\n", result); // stampa il risultato
14 } // termina la funzione main
  
```
- 2.6 a) Errore: `&number`.
 Correzione: eliminare il simbolo `&`. Discuteremo le eccezioni in seguito.
 b) Errore: `number2` non ha un `&`.
 Correzione: `number2` deve essere `&number2`. In seguito nel testo esamineremo eventuali eccezioni.
 c) Errore: il punto e virgola dopo la parentesi destra della condizione nell'istruzione `if`. L'istruzione `puts` sarà eseguita a prescindere dal fatto che la condizione nell'istruzione `if` sia vera. Il punto e virgola dopo la parentesi destra è considerato un'istruzione vuota, un'istruzione che non fa niente.
 Correzione: rimuovere il punto e virgola dopo la parentesi destra.

- d) Errore: => non è un operatore nel linguaggio C.
 Correzione: l'operatore relazionale => va cambiato in >= (maggiore o uguale a).

Esercizi

2.7 Identificate e correggete gli errori in ciascuna delle seguenti istruzioni. (Nota: potrebbe esserci più di un errore per istruzione.)

- a) `scanf("d", value);`
- b) `printf("The product of %d and %d is %d\n", x, y);`
- c) `firstNumber + secondNumber = sumOfNumbers`
- d) `if (number => largest) {`
 `largest == number;`
`}`
- e) `/* Programma per il calcolo del maggiore di tre interi */`
- f) `Scanf("%d", anInteger);`
- g) `printf("Remainder of %d divided by %d is\n", x, y, x % y);`
- h) `if (x = y); {`
 `printf(%d is equal to %d\n", x, y);`
`}`
- i) `print("The sum is %d\n", x + y);`
- j) `Printf("The value you entered is: %d\n", &value);`

2.8 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- a) _____ sono usati per documentare un programma e migliorarne la leggibilità.
- b) La funzione usata per stampare informazioni sullo schermo è _____.
- c) Un'istruzione in C che prende una decisione è _____.
- d) I calcoli sono normalmente eseguiti da istruzioni _____.
- e) La funzione _____ legge valori da tastiera.

2.9 Scrivete una singola istruzione o riga in C per eseguire ognuna delle seguenti operazioni.

- a) Stampate il messaggio "Enter two numbers.".
- b) Assegnate il prodotto delle variabili b e c alla variabile a.
- c) Dichiaretate che un programma esegue il calcolo di una retribuzione (ossia usate un testo che serva a documentare un programma).
- d) Leggete tre valori interi dalla tastiera e poneteli nelle variabili intere a, b e c.

2.10 Stabilite quali delle seguenti affermazioni sono *vere* e quali sono *false*. Se *false*, spiegiate la vostra risposta.

- a) Gli operatori del C sono calcolati da sinistra a destra.
- b) I seguenti sono tutti nomi di variabili validi: `_under_bar_, m928134, t5, j7, her_sales, his_account_total, a, b, c, z, z2.`
- c) L'istruzione `printf("a = 5;");` è un tipico esempio di istruzione di assegnazione.
- d) Un'espressione aritmetica non contenente parentesi è calcolata da sinistra a destra.
- e) I seguenti sono tutti nomi di variabili non validi: `3g, 87, 67h2, h22, 2h.`

2.11 Riempite gli vuoti in ognuna delle seguenti frasi.

- a) Quali operazioni aritmetiche sono allo stesso livello di precedenza della moltiplicazione? _____.
- b) Quando le parentesi sono annidate, quale insieme di parentesi è calcolato per primo in un'espressione aritmetica? _____.
- c) Una locazione di memoria di un computer che può contenere valori differenti nei vari momenti dell'esecuzione di un programma è chiamata una _____.

2.12 Cosa viene stampato quando viene eseguita ognuna delle seguenti istruzioni? Se non viene stampato nulla, allora rispondete "Nulla". Ponete x = 2 e y = 3.

- a) `printf("%d", x);`
- b) `printf("%d", x + x);`
- c) `printf("%s", "x=");`
- d) `printf("x=%d", x);`
- e) `printf("%d = %d", x + y, y + x);`
- f) `z = x + y;`
- g) `scanf("%d%d", &x, &y);`
- h) `// printf("x + y = %d", x + y);`
- i) `printf("%s", "\n");`

- 2.13 Quali tra le seguenti istruzioni in C contengono variabili i cui valori vengono rimpiazzati?
- a) `scanf("%d%d%d%d", &b, &c, &d, &e, &f);`
 - b) `p = i + j + k + 7;`
 - c) `printf("%s", "Values are replaced");`
 - d) `printf("%s", "a = 5");`
- 2.14 Data l'equazione $y = ax^3 + 7$, quali tra le seguenti istruzioni in C sono corrette per questa equazione?
- a) `y = a * x * x * x + 7;`
 - b) `y = a * x * x * (x + 7);`
 - c) `y = (a * x) * x * (x + 7);`
 - d) `y = (a * x) * x * x + 7;`
 - e) `y = a * (x * x * x) + 7;`
 - f) `y = a * x * (x * x + 7);`
- 2.15 Stabilite l'ordine di calcolo degli operatori in ciascuna delle seguenti istruzioni in C e determinate il valore di x dopo l'esecuzione di ogni istruzione.
- a) `x = 7 + 3 * 6 / 2 - 1;`
 - b) `x = 2 % 2 + 2 * 2 - 2 / 2;`
 - c) `x = (3 * 9 * (3 + (9 * 3 / (3))));`
- 2.16 (*Aritmetica*) Scrivete un programma che legga due interi inseriti dall'utente, quindi ne stampi la somma, il prodotto, la differenza, il quoziente e il resto.
- 2.17 (*Stampa di valori con printf*) Scrivete un programma che stampi sulla stessa riga i numeri da 1 a 4. Scrivete il programma usando i seguenti metodi.
- a) Uso di una sola istruzione `printf` senza specifiche di conversione.
 - b) Uso di una sola istruzione `printf` con quattro specifiche di conversione.
 - c) Uso di quattro istruzioni `printf`.
- 2.18 (*Confronto di interi*) Scrivete un programma che legga due interi inseriti dall'utente e quindi stampi il numero maggiore seguito dalle parole "is larger". Se i numeri sono uguali, stampate il messaggio "These numbers are equal". Usate solamente la forma a selezione singola dell'istruzione `if` che avete imparato in questo capitolo.
- 2.19 (*Aritmetica, valore maggiore e valore minore*) Scrivete un programma che riceva in ingresso tre diversi interi dalla tastiera, poi stampi la somma, la media, il prodotto, il minore e il maggiore di questi numeri. Usate solamente la forma a selezione singola dell'istruzione `if` che avete imparato in questo capitolo. Il dialogo sullo schermo deve apparire come segue:

```
Enter three different integers: 13 27 14
Sum is 54
Average is 18
Product is 4914
Smallest is 13
Largest is 27
```

2.20 (Area, diametro e circonferenza di un cerchio) Stampate il diametro, la circonferenza e l'area di un cerchio con un raggio pari a 2. Usate il valore 3,14159 per π . Usate le seguenti formule (r è il raggio): $diametro = 2r$, $circonferenza = 2\pi r$ e $area = \pi r^2$. Effettuate ognuno di questi calcoli all'interno dell'istruzione `printf` e usate la specifica di conversione `%f`. In questo capitolo abbiamo esaminato soltanto le costanti e le variabili intere. Nel Capitolo 3 esamineremo i numeri in virgola mobile, cioè i valori che possono avere punti decimali.

2.21 Che cosa stampa il seguente codice?

```
printf("%s", "*\n**\n***\n****\n*****\n");
```

2.22 (Pari o dispari) Scrivete un programma che legga un intero e determini e stampi se sia dispari o pari. Usate l'operatore di resto. Un numero pari è un multiplo di due. Qualunque multiplo di due lascia un resto di zero quando è diviso per due.

2.23 (Multipli) Scrivete un programma che legga due interi e determini e stampi se il primo è un multiplo del secondo. Usate l'operatore di resto.

2.24 Distinguete tra i termini errore fatale (o irreversibile) ed errore non fatale (non irreversibile). Perché potreste preferire di incorrere in un errore fatale piuttosto che in un errore non fatale?

2.25 (Valore intero di un carattere) Uno sguardo in avanti. In questo capitolo avete appreso qualcosa sugli interi e sul tipo `int`. Il C può anche rappresentare lettere maiuscole, lettere minuscole e una considerevole varietà di simboli speciali. Il C usa internamente numeri interi per rappresentare i singoli caratteri. L'insieme dei caratteri che un computer usa assieme alle corrispondenti rappresentazioni intere per quei caratteri si dice insieme dei caratteri di quel computer. Potete stampare l'intero equivalente della lettera maiuscola A, per esempio, eseguendo l'istruzione

```
printf("%d", 'A');
```

Scrivete un programma in C che stampi gli equivalenti interi di alcune lettere maiuscole, minuscole, di alcune cifre e simboli speciali. Determinate almeno gli equivalenti interi dei seguenti caratteri: A B C a b c 0 1 2 \$ * + / e il carattere di spazio.

2.26 (Separazione delle cifre di un intero) Scrivete un programma che riceva in ingresso un numero di cinque cifre, separi il numero nelle sue cifre individuali e stampi le cifre ciascuna separata dall'altra da tre spazi. [Suggerimento: usate combinazioni di divisioni intere con l'operazione di resto.] Per esempio, se l'utente scrive 42139, il programma deve stampare

4	2	1	3	9
---	---	---	---	---

2.27 (Tavola di quadrati e di cubi) Usando solo le tecniche che avete appreso in questo capitolo, scrivete un programma che calcoli i quadrati e i cubi dei numeri da 0 a 10 e usi tabulazioni per stampare la seguente tabella di valori:

numero	quadrato	cubo
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

2.28 (Calcolare la frequenza cardiaca) Mentre fate esercizi fisici, potete usare un cardiofrequenzimetro per controllare se la vostra frequenza cardiaca stia entro un intervallo di sicurezza indicato dai vostri istruttori e medici. Secondo l'American Heart Association (AHA) (<http://bit.ly/AHATargetHeartRates>), la formula per calcolare la vostra massima frequenza cardiaca in battiti al minuto è 220 meno la vostra età. La frequenza cardiaca ideale durante l'allenamento dovrebbe essere il 50-85% della massima frequenza cardiaca. Scrivete un programma che richieda e legga l'età dell'utente e poi calcoli e visualizzi la massima frequenza cardiaca dell'utente e l'intervallo della frequenza cardiaca ideale dell'utente. [Queste formule sono stime fornite dalla AHA; la frequenza cardiaca massima e quella sotto sforzo possono variare in base allo stato di salute, alla forma fisica e al genere di un individuo. Prima di iniziare o modificare un programma di allenamento è sempre bene consultare un medico o un operatore sanitario qualificato.]

2.29 (Ordinare in ordine crescente) Scrivete un programma che riceva in ingresso tre diversi numeri dall'utente. Stampate i numeri in ordine crescente. Ricordate che il corpo di un'istruzione `if` può contenere più di un'istruzione. Verificate che il vostro script funzioni eseguendolo su tutti e sei i possibili ordinamenti dei tre numeri. Il vostro script funziona anche su numeri uguali? [Questo esercizio è impegnativo. Nei prossimi capitoli vedremo come farlo in maniera più semplice e con molti più numeri.]

CAPITOLO

3

Sommario del capitolo

- 3.1 Introduzione
- 3.2 Algoritmi
- 3.3 Pseudocodice
- 3.4 Strutture di controllo
- 3.5 Istruzione di selezione if
- 3.6 Istruzione di selezione if...else
- 3.7 Istruzione di iterazione while
- 3.8 Formulare algoritmi, caso pratico 1: iterazione controllata da contatore
- 3.9 Formulare algoritmi con affinamento graduale top-down, caso pratico 2: iterazione controllata da sentinella
- 3.10 Formulare algoritmi con affinamento graduale top-down, caso pratico 3: istruzioni di controllo annidate
- 3.11 Operatori di assegnazione
- 3.12 Operatori di incremento e di decremento
- 3.13 Programmazione sicura in C
- 3.14 Riepilogo

Sviluppo di un programma strutturato

Obiettivi

- Usare le tecniche fondamentali per la risoluzione di problemi.
- Sviluppare algoritmi attraverso un processo top-down di affinamenti successivi.
- Selezionare le azioni da eseguire in base a una condizione utilizzando le istruzioni di selezione if e if...else.
- Eseguire ripetutamente istruzioni in un programma usando l'istruzione di iterazione while.
- Usare l'iterazione controllata da contatore e l'iterazione controllata da "sentinella".
- Usare tecniche di programmazione strutturata.
- Usare operatori di incremento, di decremento e di assegnazione.
- Continuare la nostra presentazione della programmazione sicura in C.

3.1 Introduzione

Prima di scrivere un programma per risolvere un problema particolare, dobbiamo avere una piena comprensione del problema e pianificare con cura un approccio alla soluzione. I Capitoli 3 e 4 analizzano lo sviluppo di programmi strutturati per computer. Nel Paragrafo 4.11 riassumeremo le tecniche di programmazione strutturata sviluppate qui e nel Capitolo 4.

3.2 Algoritmi

La soluzione di qualsiasi problema di calcolo implica l'esecuzione di una serie di azioni in un ordine specifico. Un **algoritmo** è una **procedura** per risolvere un problema in termini di

1. **azioni** da eseguire;
2. **ordine** in cui queste azioni vanno eseguite.

L'esempio seguente dimostra quanto sia importante specificare correttamente l'ordine di esecuzione delle azioni.

Considerate “l'algoritmo sveglia!” eseguito da un giovane dirigente d'azienda per alzarsi dal letto e andare al lavoro:

1. alzarsi dal letto;
2. togliersi il pigiama;
3. farsi una doccia;

4. vestirsi;
5. fare colazione;
6. andare al lavoro con il servizio di carpooling.

Queste azioni di routine gli consentono di arrivare al lavoro ben preparato per prendere decisioni cruciali. Supponete che gli stessi passi siano compiuti con un ordine leggermente diverso:

1. alzarsi dal letto;
2. togliersi il pigiama;
3. vestirsi;
4. farsi una doccia;
5. fare colazione;
6. andare al lavoro con il servizio di carpooling.

In questo caso il nostro giovane dirigente d'azienda si presenterebbe al lavoro bagnato fradicio. La procedura con cui si specifica l'ordine in cui le istruzioni vanno eseguite nel programma di un computer si dice **controllo del programma**. In questo e nel prossimo capitolo esamineremo le funzionalità di controllo dei programmi in C.

✓ Autovalutazione

1. (*Completere*) Una procedura per la risoluzione di un problema in termini di azioni da eseguire e ordine in cui queste azioni vanno eseguite è chiamata _____.

Risposta: algoritmo.

3.3 Pseudocodice

Lo **pseudocodice** è un linguaggio artificiale e informale, simile al linguaggio di ogni giorno, utile per lo sviluppo di algoritmi che saranno convertiti in programmi strutturati in C. Lo pseudocodice è comodo e semplice per l'utente e vi aiuta a "riflettere bene" su un programma prima di scriverlo in un linguaggio di programmazione. Lo pseudocodice non può essere eseguito su computer.

Lo pseudocodice consiste puramente di caratteri, pertanto potete scriverlo usando un comune editor di testi. In molti casi, per convertire un programma in pseudocodice preparato con cura in un corrispondente programma in C basta semplicemente sostituire le istruzioni dello pseudocodice con le loro equivalenti in C.

Lo pseudocodice descrive le *azioni* e le *decisioni* che verranno eseguite dopo la conversione dello pseudocodice in C e l'esecuzione del programma. Le definizioni non sono istruzioni eseguibili, ma semplicemente messaggi per il compilatore. Per esempio, la definizione

```
int i = 0;
```

dice al compilatore il tipo della variabile *i*, lo istruisce a riservare spazio nella memoria per la variabile e la inizializza a 0. Ma questa definizione non provoca alcuna azione quando il programma viene eseguito, come un'operazione di input o di output, oppure un calcolo o un confronto. Quindi, alcuni programmatore non includono le definizioni nel loro pseudocodice, mentre altri scelgono di elencare ogni variabile e di accennarne brevemente lo scopo.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) Lo pseudocodice è utile per lo sviluppo di algoritmi che saranno convertiti in programmi strutturati in C.
- b) Lo pseudocodice è un linguaggio di programmazione più sintetico rispetto al C.
- c) Lo pseudocodice consiste puramente di caratteri, così potete comodamente scriverlo usando un comune editor di testi.

- d) Lo pseudocodice descrive le azioni e le decisioni che vengono eseguite dopo la conversione del programma in linguaggio C e la sua esecuzione.

Risposta: b) è falsa. Lo pseudocodice non è un vero linguaggio di programmazione per computer, ma vi può aiutare a “riflettere bene” su un programma prima di scriverlo in un linguaggio di programmazione.

3.4 Strutture di controllo

Normalmente le istruzioni in un programma sono eseguite una dopo l'altra nell'ordine in cui sono scritte. Questa semplice modalità è chiamata **esecuzione sequenziale**. Varie istruzioni in C che presto esamineremo vi permetteranno di specificare come istruzione successiva da eseguire una diversa da quella successiva nella sequenza. Questa modalità è chiamata **trasferimento del controllo**.

Durante gli anni Sessanta divenne chiaro che l'uso indiscriminato del trasferimento del controllo stava alla radice di una grande quantità di problemi incontrati dagli sviluppatori del software. Il dito era puntato contro l'**istruzione goto**, che permette di specificare un trasferimento del controllo a uno dei tanti punti possibili in un programma. Il principio della cosiddetta programmazione strutturata divenne quasi sinonimo di “**eliminazione del goto**”.

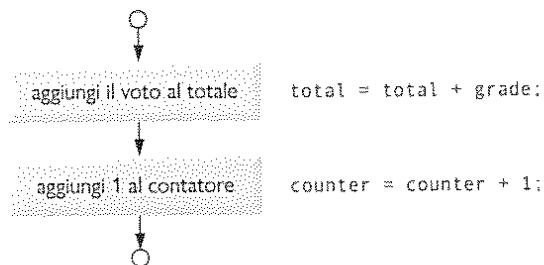
La ricerca di Böhm e Jacopini¹ ha dimostrato che i programmi potrebbero essere scritti senza istruzioni goto. Per i programmatori la sfida dell'epoca era quella di spostare i loro stili verso la “programmazione senza goto”. Solo negli anni Settanta la pratica professionale della programmazione ha iniziato a prendere sul serio la programmazione strutturata. I risultati sono stati impressionanti, in quanto gli sviluppatori del software riportavano tempi di sviluppo ridotti, una consegna più frequente dei sistemi nei tempi stabiliti e un più frequente completamento dei progetti software all'interno dei budget. I programmi prodotti con tecniche strutturate erano più chiari, più facili da correggere e da modificare e, con buona probabilità, senza errori sin dal primo momento.

Il lavoro di Böhm e Jacopini ha dimostrato che tutti i programmi potrebbero essere scritti in termini di sole tre **strutture di controllo**, ossia la **struttura sequenziale**, la **struttura di selezione** e la **struttura di iterazione**. La struttura sequenziale è semplice: a meno che non venga specificato diversamente, il computer esegue le istruzioni in C una dopo l'altra nell'ordine in cui sono scritte.

Diagrammi di flusso

Un **diagramma di flusso** è una rappresentazione grafica di un algoritmo o di una porzione di un algoritmo. I diagrammi di flusso si disegnano usando certi simboli specifici come rettangoli, rombi, rettangoli arrotondati e cerchietti, collegati da frecce chiamate **linee di flusso**.

I diagrammi di flusso sono utili per sviluppare e rappresentare gli algoritmi, anche se la maggior parte dei programmatori preferisce lo pseudocodice. I diagrammi di flusso mostrano chiaramente come operano le strutture di controllo. Consideriamo il seguente diagramma di flusso per la struttura sequenziale della porzione di un algoritmo che calcola la media della classe in riferimento ai voti ottenuti in un quiz:



1. C. Böhm, G. Jacopini, “Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules,” *Communications of the ACM*, Vol. 9, No. 5, maggio 1966, pp. 336-371. Questo classica pubblicazione in ambito informatico si può trovare su numerosi siti online.

Il simbolo rettangolo (o simbolo di azione) indica qualsiasi azione, come un calcolo o un'operazione di input o output. Le linee di flusso indicano l'ordine nel quale vanno eseguite le azioni. Questo segmento di programma per prima cosa aggiunge `grade` (voto) a `total` (totale), poi aggiunge 1 a `counter` (contatore). Come vedremo a breve, dovunque in un programma si possa inserire una singola azione è possibile inserire diverse azioni in sequenza.

Quando si disegna un diagramma di flusso che rappresenta un algoritmo completo, il primo simbolo usato nel diagramma è il **simbolo rettangolo arrotondato** contenente la parola "Inizio"; l'ultimo simbolo usato è il simbolo rettangolo arrotondato contenente la parola "Fine". Quando si disegna solo una porzione di un algoritmo, i simboli rettangolo arrotondato sono omessi e al loro posto si usano piccoli cerchi chiamati **simboli connettori**.

Istruzioni di selezione in C

Il C fornisce tre tipi di strutture di selezione nella forma di istruzioni:

- **l'istruzione di selezione singola `if`** (Paragrafo 3.5) seleziona (esegue) un'azione (o un gruppo di azioni) solo se una condizione è vera;
- **l'istruzione di selezione doppia `if...else`** (Paragrafo 3.6) esegue un'azione (o un gruppo di azioni) se una condizione è *vera* ed esegue un'azione diversa (o un gruppo di azioni diverso) se la condizione è *falsa*;
- **l'istruzione di selezione multipla `switch`** (analizzata nel prossimo capitolo) esegue una fra varie azioni differenti, a seconda del valore di un'espressione.

Istruzioni di iterazione in C

Il C fornisce tre tipi di strutture di iterazione nella forma di istruzioni, cioè `while` (Paragrafo 3.7), `do...while` e `for`. Queste istruzioni eseguono compiti ripetutamente. Analizzeremo `do...while` e `for` nel prossimo capitolo.

Riepilogo delle istruzioni di controllo

Questo è tutto quello che c'è. Il linguaggio C ha soltanto sette istruzioni di controllo: sequenza, tre tipi di selezione e tre tipi di iterazione. Ogni programma in C è formato dalla combinazione di tante istruzioni di controllo di ognuno di questi tipi quante sono quelle appropriate per l'algoritmo che il programma implementa.

Vedremo che la rappresentazione a diagramma di flusso di ogni istruzione di controllo ha due simboli cerchietto, uno nel punto di ingresso dell'istruzione di controllo e uno nel punto di uscita. Queste **istruzioni di controllo a un solo ingresso e a una sola uscita** facilitano la costruzione di programmi chiari.

I segmenti di diagrammi di flusso relativi alle istruzioni di controllo possono essere attaccati l'uno all'altro collegando il punto di uscita di un'istruzione di controllo al punto di ingresso della successiva. Ciò è molto simile al modo in cui un bambino mette l'uno sull'altro i blocchi di costruzioni, per cui chiamiamo quest'operazione **accatastamento delle istruzioni di controllo**. Apprenderemo nel seguito del capitolo che vi è soltanto un altro modo per connettere le istruzioni di controllo: l'annidamento. Qualsiasi programma in C che avremo mai bisogno di costruire potrà quindi essere costruito partendo soltanto da sette tipi differenti di istruzioni di controllo combinate in due soli modi. È l'essenza della semplicità.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- Normalmente, le istruzioni in un programma vengono eseguite una dopo l'altra nell'ordine in cui sono scritte. Questo è chiamato trasferimento del controllo.
- I programmi possono essere scritti senza istruzioni `goto`.
- Tutti i programmi potrebbero essere scritti in termini di sole tre strutture di controllo, ossia la struttura sequenziale, la struttura di selezione e la struttura di iterazione.
- La struttura sequenziale è semplice: a meno che non venga specificato diversamente, il computer esegue le istruzioni in C una dopo l'altra nell'ordine in cui sono scritte.

Risposta: a) è *falsa*. In realtà, questa modalità è chiamata esecuzione sequenziale.

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) Il linguaggio C ha soltanto sette istruzioni di controllo: sequenza, tre tipi di selezione e tre tipi di iterazione.
- b) Istruzioni di controllo a un solo ingresso e a una sola uscita facilitano la costruzione di programmi chiari.
- c) Il collegamento del punto di uscita di un'istruzione di controllo al punto di ingresso della successiva viene chiamato accatastamento delle istruzioni di controllo.
- d) L'unico altro modo per collegare le istruzioni di controllo è il metodo della nidificazione. Qualsiasi programma in C che avremo mai bisogno di costruire potrà quindi essere costruito partendo soltanto da sette tipi differenti di istruzioni di controllo combinate in due soli modi.

Risposta: d) è *falsa*. L'unico altro modo per collegare le istruzioni di controllo è il metodo dell'annidamento.

3.5 Istruzione di selezione if

Le istruzioni di selezione sono usate per scegliere tra linee di azione alternative. Per esempio, supponiamo che il voto minimo per superare un esame sia 60. La seguente istruzione in pseudocodice verifica se la condizione "il voto dello studente è maggiore o uguale a 60" è *vera* o *falsa*:

Se il voto dello studente è maggiore o uguale a 60
Stampa "Passed"

Se la condizione è *vera*, allora viene stampato "Passed" e viene "eseguita" nell'ordine la successiva istruzione nello pseudocodice. Ricordate che lo pseudocodice non è un vero linguaggio di programmazione. Se la condizione è *falsa*, la stampa viene ignorata e viene eseguita nell'ordine la successiva istruzione nello pseudocodice.

Il precedente pseudocodice è scritto in C come

```
if (grade >= 60) {
    puts("Passed");
} // fine di if
```

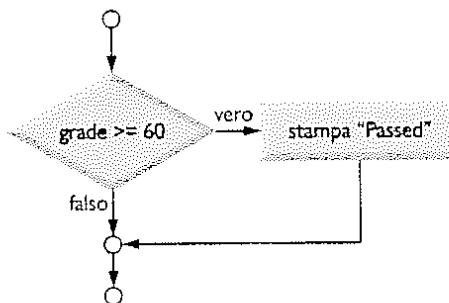
Ovviamente, dovete dichiarare la variabile grade come int, ma il codice dell'istruzione if in C corrisponde strettamente allo pseudocodice. Questa è una delle proprietà dello pseudocodice che lo rende uno strumento utile per lo sviluppo di un programma.

Indentazione dell'istruzione if

L'indentazione nella seconda riga dell'istruzione if è facoltativa ma altamente raccomandata, in quanto consente di mettere in rilievo la struttura propria dei programmi strutturati. Il compilatore ignora i **caratteri di spaziatura** come spazi, tabulazioni e newline usati per l'indentazione e la spaziatura verticale.

Diagramma di flusso dell'istruzione if

Il seguente segmento di un diagramma di flusso illustra l'istruzione if a selezione singola:



Esso contiene quello che è forse il simbolo più importante del diagramma, il **rombo** (o **simbolo di decisione**), il quale indica che deve essere presa una decisione. Il simbolo di decisione contiene un'espressione, come una condizione, che può essere *vera* o *falsa*. Le due linee di flusso che emergono da esso indicano le direzioni da prendere quando l'espressione è *vera* o *falsa*. Una decisione può basarsi su qualsiasi espressione: se l'espressione ha valore uguale a zero è trattata come *falsa* e se ha valore diverso da zero è trattata come *vera*.

L'istruzione **if** è un'istruzione a un solo ingresso e a una sola uscita. Presto apprenderemo che anche il segmento del diagramma di flusso per le rimanenti strutture di controllo può contenere simboli rettangolo per indicare le azioni da eseguire e simboli rombo per indicare le decisioni da prendere. Questo è il modello di programmazione azione/decisione cui abbiamo dato risalto.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) L'istruzione in pseudocodice

Se il voto dello studente è maggiore o uguale a 60
Stampa "Passed"

può essere scritta in C come

```
if (grade >= 60) {
    puts("Passed");
} // fine di if
```

- b) Le due linee di flusso che emergono dal simbolo di decisione del diagramma di flusso indicano le direzioni da prendere quando l'espressione contenuta nel simbolo è *vera* oppure *falsa*.
- c) Le decisioni possono basarsi su qualsiasi espressione: se l'espressione ha valore diverso da zero è trattata come *falsa* e se ha valore uguale a zero è trattata come *vera*.
- d) L'istruzione **if** è un'istruzione a un solo ingresso e a una sola uscita.

Risposta: c) è *falsa*. In realtà, le decisioni possono basarsi su qualsiasi espressione: se l'espressione ha valore uguale a zero è trattata come *falsa* e se ha valore diverso da zero è trattata come *vera*.

3.6 Istruzione di selezione **if...else**

L'istruzione di selezione **if...else** permette di specificare che vanno eseguite azioni differenti quando la condizione è *vera* e quando è *falsa*. Per esempio, l'istruzione nello pseudocodice

Se il voto dello studente è maggiore o uguale a 60

Stampa "Passed"

altrimenti

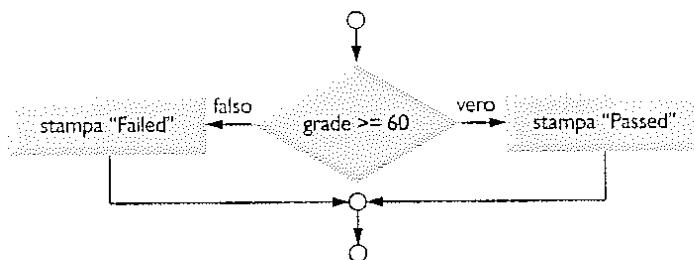
Stampa "Failed"

stampa "Passed" se il voto dello studente è maggiore o uguale a 60, altrimenti, stampa "Failed." In entrambi i casi, dopo la stampa, viene "eseguita" la successiva istruzione in sequenza nello pseudocodice. Anche il corpo dell'*altrimenti* viene indentato. Se vi sono diversi livelli di indentazione in un programma, ciascuno di essi deve essere indentato con la stessa quantità di spazio. Il precedente pseudocodice può essere scritto in C come

```
if (grade >= 60) {
    puts("Passed");
} // fine di if
else {
    puts("Failed");
} // fine di else
```

Diagramma di flusso dell'istruzione if...else

Il seguente diagramma di flusso illustra il flusso di controllo nell'istruzione if...else:



Espressioni condizionali

L'**operatore condizionale** (`? :`) è strettamente correlato all'istruzione if...else. Si tratta dell'unico **operatore ternario** in C, ovvero si applica a tre operandi. Un operatore condizionale e i suoi tre operandi formano un'**espressione condizionale**. Il primo operando è una condizione. Il secondo operando è il valore dell'espressione condizionale se la condizione è *vera*. Il terzo è il valore dell'espressione condizionale se la condizione è *falsa*. Per esempio, l'espressione condizionale contenuta come argomento nella seguente istruzione puts assume come valore la stringa "Passed" se la condizione grade ≥ 60 è vera e la stringa "Failed" se la condizione è falsa:

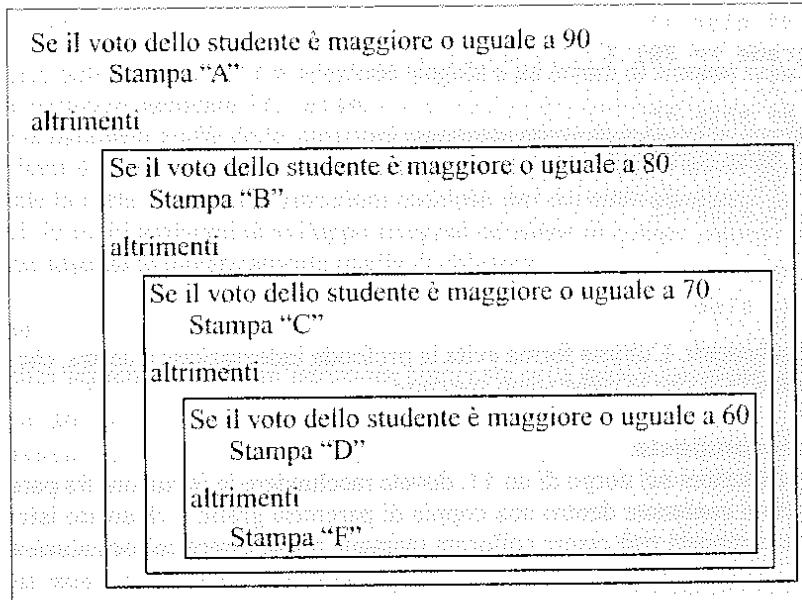
```
puts((grade >= 60) ? "Passed" : "Failed");
```

Gli operatori condizionali possono essere usati in punti dove non è possibile ricorrere all'istruzione if...else, includendo espressioni e argomenti a funzioni (come printf). Utilizzate espressioni dello stesso tipo per il secondo e il terzo operando dell'operatore condizionale (`? :`) per evitare errori insidiosi.



Istruzioni annidate if...else

Le **istruzioni annidate if...else** effettuano test su casi multipli attraverso l'inserimento di istruzioni if...else all'interno di istruzioni if...else. Per esempio, la seguente istruzione in pseudocodice stamperà A per i voti dell'esame maggiori o uguali a 90, B per i voti maggiori o uguali a 80 (ma minori di 90), C per i voti maggiori o uguali a 70 (ma minori di 80), D per i voti maggiori o uguali a 60 (ma minori di 70) e F per tutti gli altri voti.



Questo pseudocodice può essere scritto in C come:

```

if (grade >= 90) {
    puts("A");
} // fine di if
else {
    if (grade >= 80) {
        puts("B");
    } // fine di if
    else {
        if (grade >= 70) {
            puts("C");
        } // fine di if
        else {
            if (grade >= 60) {
                puts("D");
            } // fine di if
            else {
                puts("F");
            } // fine di else
        } // fine di else
    } // fine di else
} // fine di else

```

Se la variabile grade è maggiore o uguale a 90, tutte e quattro le condizioni saranno vere, ma soltanto l'istruzione puts dopo il primo test sarà eseguita. Poi, la parte else dell'istruzione if...else "esterna" viene saltata, bypassando il resto dell'istruzione if...else annidata.

La maggior parte dei programmatori scrive la precedente istruzione if come

```

if (grade >= 90) {
    puts("A");
} // fine di if
else if (grade >= 80) {
    puts("B");
} // fine di else if
else if (grade >= 70) {
    puts("C");
} // fine di else if
else if (grade >= 60) {
    puts("D");
} // fine di else if
else {
    puts("F");
} // fine di else

```

Le due forme sono equivalenti. L'ultima forma evita la profonda indentazione a destra, che riduce la leggibilità del programma e spesso fa sì che le righe siano spezzate.

Blocchi e istruzioni composte

Per includere diverse istruzioni nel corpo di un if, dovete racchiudere le istruzioni fra parentesi graffe ({ e }). L'insieme delle istruzioni contenute dentro una coppia di parentesi graffe è chiamato **istruzione composta** o **blocco**. Un'istruzione composta può essere collocata ovunque possa essere collocata una singola istruzione in un programma.

La parte `else` della seguente istruzione `if...else` include un'istruzione composta contenente due istruzioni da eseguire se la condizione è *falsa*:

```
if (grade >= 60) {
    puts("Passed.");
} // fine di if
else {
    puts("Failed.");
    puts("You must take this course again.");
} // fine di else
```

Se il voto (`grade`) è inferiore a 60, il programma esegue entrambe le istruzioni `puts` nel corpo dell'`else` e stampa:

```
Failed.
You must take this course again.
```

Le parentesi graffe attorno alle due istruzioni nella clausola `else` sono importanti. Senza di esse l'istruzione

```
    puts("You must take this course again.");
```

sarebbe al di fuori del corpo della parte `else` (e al di fuori dell'istruzione `if...else`) e verrebbe eseguita a prescindere dal fatto che il voto sia inferiore o meno a 60, quindi anche uno studente che supera la prova dovrebbe ripetere il corso. Per evitare problemi come questo, includete sempre il corpo delle istruzioni di controllo tra parentesi graffe (`{ e }`), anche se contiene solo una singola istruzione. Questo risolve il problema dell'`else` sospeso, che analizzeremo negli esercizi di questo capitolo.

Tipi di errori

-  Un errore di sintassi (come `"else"` scritto sbagliato) viene individuato dal compilatore. Un errore logico ha il suo effetto al momento dell'esecuzione. Un errore logico irreversibile fa sì che il programma fallisca e termini prematuramente. Un errore logico non irreversibile permette al programma di continuare l'esecuzione ma gli fa produrre risultati scorretti.

Istruzione vuota

-  Ovunque sia possibile collocare un'istruzione singola o un'istruzione composta è anche possibile collocare un'istruzione vuota, rappresentata da un punto e virgola (`;`). L'inserimento di un punto e virgola dopo una condizione `if`, come in

```
if (grade >= 60);
```

provoca un errore logico nelle istruzioni `if` a selezione singola e un errore di sintassi nelle istruzioni `if` a selezione doppia e nelle istruzioni annidate `if...else`.

Digitate entrambe le parentesi graffe delle istruzioni composte prima di scrivere le singole istruzioni al loro interno. Questo vi eviterà di omettere una o entrambe le parentesi, prevenendo errori di sintassi (come un'istruzione `if` nella quale la parte `if` contiene istruzioni multiple, per cui sono necessarie entrambe le parentesi graffe) ed errori logici. In molti ambienti di sviluppo integrati ed editor di codice, quando digitate la parentesi graffa di apertura viene inserita automaticamente quella di chiusura.

✓ Autovalutazione

1. *(Vero/Falso)* Il codice seguente include un'istruzione composta nella parte `else` di un'istruzione `if...else`:

```
if (grade >= 60) {
    puts("Passed.");
} // fine di if
else
    puts("Failed.");
    puts("You must take this course again.");
```

Risposta: *Falso.* Mancano le parentesi graffe intorno alle due istruzioni `puts` nella parte `else` dell'istruzione `if...else`. Il codice corretto con l'istruzione composta è:

```
if (grade >= 60) {
    puts("Passed.");
} // fine di if
else {
    puts("Failed.");
    puts("You must take this course again.");
} // fine di else
```

2. (*Vero/Falso*) L'espressione condizionale contenuta come argomento nella seguente istruzione `puts`

```
puts((grade >= 60) : "Passed" ? "Failed");
```

assume come valore la stringa "Passed" se la condizione `grade >= 60` è *vera* e la stringa "Failed" se la condizione è *falsa*.

Risposta: *Falso.* Questa istruzione non può essere compilata perché `? e :` dell'operatore condizionale sono invertiti. L'istruzione corretta per ottenere il risultato desiderato è la seguente:

```
puts((grade >= 60) ? "Passed" : "Failed");
```

3.7 Istruzione di iterazione while

Un'**istruzione di iterazione** (chiamata anche **istruzione di ripetizione** o **ciclo**) permette di specificare che un'azione va ripetuta finché una qualche condizione rimane vera. L'istruzione in pseudocodice

Finché ci sono ancora elementi nella mia lista della spesa

Acquista l'elemento successivo e cancellalo dalla mia lista

descrive un'iterazione che avviene quando si fa la spesa. La condizione "ci sono ancora elementi nella mia lista della spesa" può essere *vera* o *falsa*. Se è *vera*, l'acquirente esegue l'azione "Acquista l'elemento successivo e cancellalo dalla mia lista". Questa azione sarà eseguita ripetutamente finché la condizione rimane *vera*. Alla fine, la condizione diventerà *falsa* (quando l'ultimo elemento nella lista della spesa sarà stato acquistato e cancellato dalla lista). A questo punto l'iterazione terminerà e verrà "eseguita" la prima istruzione nello pseudocodice dopo l'istruzione di iterazione.

Calcolare la prima potenza di 3 maggiore di 100

Come esempio di un'istruzione `while`, considerate un segmento di programma scritto per trovare la prima potenza di 3 maggiore di 100. Supponete che la variabile intera `product` sia stata inizializzata a 3. Quando il seguente codice terminerà l'esecuzione, `product` conterrà la risposta desiderata:

```
int product = 3;

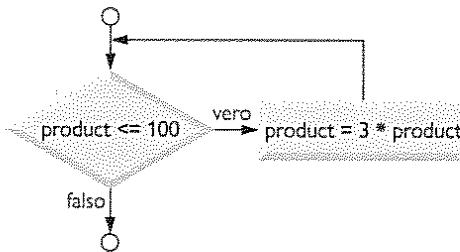
while (product <= 100) {
    product = 3 * product;
}
```

Il ciclo moltiplica ripetutamente per 3 la variabile `product`, che assume in successione i valori 9, 27 e 81. Quando `product` diventa 243, la condizione `product <= 100` diventa *falsa*, terminando l'iterazione: il valore finale di `product` è 243. L'esecuzione del programma prosegue con l'istruzione successiva dopo `while`. È necessario inserire nel corpo dell'istruzione `while` un'azione che faccia sì che alla fine la condizione diventi *falsa*, altrimenti il ciclo non terminerà mai: un errore logico chiamato **ciclo infinito**. Le istruzioni contenute nell'istruzione di iterazione `while` ne costituiscono il **corpo**, che può essere un'istruzione singola o un'istruzione composta.



Diagramma di flusso dell'istruzione while

Il seguente diagramma di flusso illustra la precedente istruzione di iterazione while:



✓ Autovalutazione

- (Segmento di programma)** Il segmento di programma relativo all'istruzione while di questa sezione trova la prima potenza di 3 maggiore di 100. Riscrivete questo segmento di programma in modo che trovi la prima potenza di 2 maggiore o uguale a 1024, lasciando il valore in *product*.

Risposta:

```

int product = 2;

while (product < 1024) {
    product = 2 * product;
}
  
```

- (Completare)** Un'azione nel corpo di un'istruzione while deve necessariamente far sì che alla fine la condizione diventi *falsa*, altrimenti il ciclo non terminerà mai. Questo è un errore logico chiamato _____.

Risposta: ciclo infinito.

3.8 Formulare algoritmi, caso pratico 1: iterazione controllata da contatore

Per illustrare come vengono sviluppati gli algoritmi, cerchiamo le soluzioni per due varianti del problema della media di una classe in questo paragrafo e nel prossimo. Considerate l'istruzione del seguente problema:

Una classe di dieci studenti fa un quiz. Avete a disposizione i voti (numeri interi compresi tra 0 e 100) per questo quiz. Determinate la media della classe in riferimento al quiz.

La media della classe è uguale alla somma dei voti divisa per il numero degli studenti. L'algoritmo per risolvere questo problema deve ricevere in ingresso i voti, eseguire il calcolo della media e stampare il risultato.

Pseudocodice per il problema della media di una classe

Usiamo lo pseudocodice per elencare le azioni da eseguire e specificare l'ordine in cui devono essere eseguite. Usiamo l'**iterazione controllata da contatore** per inserire i voti uno alla volta. Questa tecnica usa una variabile chiamata **contatore** per specificare il numero delle volte in cui un insieme di istruzioni deve essere eseguito. In questo esempio, sappiamo che dieci studenti hanno fatto un quiz, quindi dobbiamo inserire 10 voti. L'iterazione termina quando il contatore supera il valore 10. In questo caso pratico, presentiamo semplicemente l'algoritmo in pseudocodice finale (Figura 3.1) e il corrispondente programma in C (Figura 3.2). Nel prossimo caso pratico, mostreremo come vengono sviluppati gli algoritmi in pseudocodice. L'iterazione controllata da

contatore è spesso chiamata **iterazione definita** perché il numero delle iterazioni si conosce prima che il ciclo inizi l'esecuzione.

- 1 Imposta il totale uguale a zero
- 1 Imposta il contatore dei voti uguale a uno
- 1
- 1 Finché il contatore dei voti è minore o uguale a dieci
 - 1 Inserisci il voto successivo
 - 1 Somma il voto al totale
 - 1 Somma uno al contatore dei voti
 - 1
- 1 Imposta la media della classe uguale al totale diviso per dieci
- 2 Stampa la media della classe

Figura 3.1 Algoritmo in pseudocodice che usa l'iterazione controllata da un contatore per risolvere il problema della media di una classe.

```

1 // fig03_02.c
2 // Media di una classe con l'iterazione controllata da contatore.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     // fase di inizializzazione
8     int total = 0; // inizializza il totale dei voti a 0
9     int counter = 1; // numero del voto da inserire successivamente
10
11    // fase di elaborazione
12    while (counter <= 10) { // ripeti 10 volte
13        printf("%s", "Enter grade: "); // prompt per l'input
14        int grade = 0; // valore del voto
15        scanf("%d", &grade); // leggi il voto
16        total = total + grade; // somma il voto al totale
17        counter = counter + 1; // incrementa il contatore
18    } // fine di while
19
20    // fase di terminazione
21    int average = total / 10; // divisione intera
22    printf("Class average is %d\n", average); // stampa il risultato
23 } // fine della funzione main

```

```

Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81

```

Figura 3.2 Problema della media di una classe con iterazione controllata da contatore.

 Un totale è una variabile (riga 8) usata per accumulare la somma di una serie di valori. Un contatore è una variabile (riga 9) usata per contare (in questo caso, per contare il numero dei voti inseriti). Le variabili usate per memorizzare i totali devono essere inizializzate a zero, altrimenti la somma includerebbe anche il precedente valore memorizzato nella locazione di memoria del totale. Devono essere inizializzati tutti i contatori e i totali. I contatori sono solitamente inizializzati a zero o a uno, a seconda del loro uso (presenteremo esempi di ognuno dei due casi). Una variabile non inizializzata contiene un valore "spazzatura", l'ultimo valore memorizzato nella locazione di memoria riservata per quella variabile. Se un contatore o un totale non viene inizializzato, i risultati del vostro programma saranno probabilmente scorretti. Questi sono esempi di errori logici.

La media del programma precedente era 81, ma la somma dei voti che abbiamo inserito era 817. Ovviamente, 817 diviso per 10 produce il valore 81,7, cioè un numero con la virgola decimale. Vedremo come trattare tali numeri (chiamati *numeri in virgola mobile*) nel prossimo paragrafo.

✓ Autovalutazione

1. (*Completare*) L'iterazione controllata da contatore è spesso chiamata iterazione _____, perché il numero delle iterazioni si conosce prima che il ciclo inizi l'esecuzione.

Risposta: definita.

2. (*Vero/Falso*) Le variabili usate per memorizzare i totali devono essere inizializzate a uno prima di essere usate in un programma, altrimenti la somma includerebbe anche il precedente valore memorizzato nella locazione di memoria del totale.

Risposta: Falso. In realtà, le variabili usate per memorizzare i totali devono essere inizializzate a zero prima di essere utilizzate in un programma, altrimenti la somma includerebbe il precedente valore memorizzato nella locazione di memoria del totale.

3.9 Formulare algoritmi con affinamento graduale top-down, caso pratico 2: iterazione controllata da sentinella

Generalizziamo il problema del calcolo della media di una classe. Consideriamo il seguente problema:

Sviluppare un programma per il calcolo della media di una classe in grado di elaborare un numero arbitrario di voti ogni volta che è in esecuzione.

Nel primo esempio del calcolo della media di una classe, sapevamo in anticipo che i voti erano 10. In questo esempio non si dà alcuna indicazione su quanti voti verranno inseriti. Il programma deve elaborare un numero arbitrario di voti. Come può il programma stabilire quando fermare l'inserimento dei voti? Come saprà quando calcolare e stampare la media della classe?

Valori sentinella

Un modo per risolvere questo problema è quello di usare un valore speciale, chiamato **valore sentinella** (o anche **valore di segnale**, **valore fittizio** o **valore di flag**), per indicare la "fine dell'inserimento dei dati". L'utente scrive i voti finché non sono stati inseriti tutti i voti validi, poi scrive il valore sentinella per indicare che l'ultimo voto è stato inserito. L'iterazione controllata da sentinella è spesso chiamata **iterazione indefinita** perché il numero di iterazioni non è noto prima che il ciclo inizi l'esecuzione.

Il valore sentinella deve essere scelto in modo tale che non possa essere confuso con un valore di input accettabile. Poiché i voti su un quiz sono numeri interi non negativi, -1 è un valore sentinella accettabile per questo problema. Quindi, un'esecuzione del programma della media di una classe potrebbe elaborare una sequenza di input come 95, 96, 75, 74, 89 e -1. Il programma poi calcolerebbe e stamperebbe la media della classe per i voti 95, 96, 75, 74 e 89. Il valore sentinella -1 non deve entrare nel calcolo della media.

Affinamento graduale top-down

Affrontiamo il problema della media di una classe con una tecnica chiamata **affinamento graduale top-down**, che è essenziale allo sviluppo di programmi ben strutturati.

Iniziamo con una rappresentazione dello pseudocodice a livello **top**:

Determina la media della classe per il quiz

Il livello top è un'istruzione singola che rappresenta la funzione complessiva del programma. Come tale il livello top è, in effetti, una rappresentazione completa di un programma. Purtroppo, il livello top raramente contiene una quantità sufficiente di dettagli per scrivere il programma in C. Così iniziamo ora il processo di affinamento. Suddividiamo il livello top in una serie di compiti più piccoli e li elenchiamo nell'ordine in cui devono essere eseguiti. Ciò risulta nel seguente **primo affinamento**.

Inizializza le variabili

Ricevi in ingresso, somma e conta i voti del quiz

Calcola e stampa la media della classe

Qui è stata usata solo la struttura sequenziale (i passi elencati vanno eseguiti in ordine, uno dopo l'altro). Ogni affinamento, come pure lo stesso livello top, è una specifica completa dell'algoritmo; varia soltanto il livello di dettaglio.

Secondo affinamento

Per procedere al successivo livello di affinamento, cioè al **secondo affinamento**, useremo alcune variabili specifiche. Avremo bisogno di:

- un totale corrente dei numeri;
- un conteggio di quanti numeri sono stati elaborati;
- una variabile che riceve il valore di ogni singolo voto quando è inserito;
- una variabile per contenere la media calcolata.

L'istruzione in pseudocodice

Inizializza le variabili

può essere affinata come segue:

Inizializza il totale a zero

Inizializza il contatore a zero

Si noti che vanno inizializzati soltanto il totale e il contatore. Le variabili per la media calcolata e il voto inserito dall'utente non necessitano di essere inizializzate perché i loro valori saranno calcolati e inseriti dall'utente, rispettivamente. L'istruzione in pseudocodice

Ricevi in ingresso, somma e conta i voti del quiz

richiede una *struttura di iterazione* che di volta in volta legge ogni voto. Poiché non sappiamo in anticipo quanti sono i voti da elaborare, useremo l'iterazione controllata da sentinella. L'utente inserirà i voti validi uno alla volta. Dopo aver scritto l'ultimo voto valido, l'utente scriverà il valore sentinella. Il programma verificherà questo valore dopo che sarà inserito ogni voto e terminerà il ciclo quando sarà inserita la sentinella. L'affinamento della precedente istruzione in pseudocodice è allora

Ricevi in ingresso il primo voto (eventualmente la sentinella)

Finché l'utente non ha ancora inserito la sentinella

 Aggiungi questo voto al totale corrente

 Aggiungi uno al contatore dei voti

Ricevi in ingresso il voto successivo (eventualmente la sentinella)

Nello pseudocodice non usiamo parentesi graffe intorno all'insieme di istruzioni che formano il corpo di un ciclo, ma semplicemente indentiamo tutte queste istruzioni sotto il *while*. Anche in questo caso, lo pseudocodice è un ausilio per lo sviluppo informale di un programma.

L'istruzione in pseudocodice

Calcola e stampa la media della classe

si può affinare come segue:

```

Se il contatore non è uguale a zero
    Imposta la media uguale al totale diviso per il contatore
    Stampa la media
altrimenti
    Stampa "No grades were entered"

```

 Siamo stati attenti a verificare la possibilità della **divisione per zero**, un **errore irreversibile** che, se non viene scoperto, provoca l'arresto del programma (in questo caso detto “crashing”, cioè “arresto anomalo”). Dovete controllare esplicitamente questo caso e trattarlo adeguatamente nel vostro programma (per esempio stampando un messaggio di errore) piuttosto che permettere che si verifichi l'errore irreversibile.

Secondo affinamento completo

Il secondo affinamento completo è mostrato nella Figura 3.3. Includiamo alcune righe completamente vuote nello pseudocodice per migliorarne la leggibilità.

```

1 Inizializza il totale a zero
1 Inizializza il contatore a zero
1
1 Ricevi in ingresso il primo voto (eventualmente la sentinella)
1 Finché l'utente non ha ancora inserito la sentinella
1     Aggiungi il voto al totale corrente
1     Aggiungi uno al contatore dei voti
2     Ricevi in ingresso il voto successivo (eventualmente la sentinella)
3
1 Se il contatore non è uguale a zero
1     Imposta la media uguale al totale diviso per il contatore
1     Stampa la media
1 altrimenti
2     Stampa "No grades were entered"

```

Figura 3.3 Algoritmo in pseudocodice che usa l'iterazione controllata da sentinella per risolvere il problema del calcolo della media di una classe.

Fasi in un programma base

Molti programmi possono essere divisi logicamente in tre fasi:

- una **fase di inizializzazione** che inizializza le variabili del programma;
- una **fase di elaborazione** che riceve in ingresso i valori dei dati e modifica conseguentemente le variabili del programma;
- una **fase di chiusura** che calcola e stampa i risultati finali.

Numeri dei livelli di affinamento dello pseudocodice

L'algoritmo in pseudocodice nella Figura 3.3 risolve il problema del calcolo della media di una classe nella sua forma più generale. Questo algoritmo è stato sviluppato dopo due soli livelli di affinamento. Talvolta sono necessari più livelli. Il processo di affinamento graduale top-down termina quando l'algoritmo in pseudocodice è specificato con sufficiente dettaglio tale da poter convertire lo pseudocodice direttamente in C.

Quando si deve risolvere un problema su un computer, la parte più difficile consiste nello sviluppare l'algoritmo per la soluzione. Una volta che è stato definito il corretto algoritmo, in genere è poi semplice produrre un programma funzionante in C. Molti scrivono i programmi senza avvalersi di strumenti come lo pseudocodice, ritenendo che l'obiettivo finale sia risolvere il problema e che la scrittura dello pseudocodice comporti soltanto una perdita di tempo nella realizzazione degli output finali. Questo può valere per programmi di dimensioni limitate, sviluppati per uso personale, ma per programmi e sistemi software importanti, come quelli su cui lavorerete in campo professionale, un processo di sviluppo formale è fondamentale.

Programma del calcolo della media di una classe per un numero arbitrario di voti

La Figura 3.4 mostra il programma in C e due esempi di esecuzione. Sebbene vengano inseriti solo voti interi, è probabile che il calcolo della media produca un numero con un punto decimale (che sostituisce la virgola nella notazione del C). Il tipo `int` non può rappresentare un tale numero. Quindi questo programma introduce il tipo di dati `double` per trattare i numeri con punto decimale (chiamati **numeri in virgola mobile**) e un operatore speciale chiamato *operatore cast* per far sì che vengano utilizzati numeri in virgola mobile nel calcolo della media. Questi argomenti sono spiegati dopo la presentazione del programma. Notate che le righe 13 e 23 includono il valore sentinella nei prompt che richiedono l'inserimento dei dati. Questa è una buona pratica di programmazione in un ciclo controllato da sentinella.

```

1 // fig03_04.c
2 // Media di una classe con iterazione controllata da sentinella.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     // fase di inizializzazione
8     int total = 0; // inizializza il totale
9     int counter = 0; // inizializza il contatore del ciclo
10
11    // fase di elaborazione
12    // ricevi il primo voto dall'utente
13    printf("%s", "Enter grade, -1 to end: "); // prompt per l'input
14    int grade = 0; // valore del voto
15    scanf("%d", &grade); // leggi il voto dall'utente
16
17    // ripeti finché non viene letto il valore sentinella
18    while (grade != -1) {
19        total = total + grade; // aggiungi il voto al totale
20        counter = counter + 1; // incrementa il contatore
21
22        // ricevi il voto successivo dall'utente
23        printf("%s", "Enter grade, -1 to end: "); // prompt per l'input
24        scanf("%d", &grade); // leggi il prossimo voto
25    } // fine di while
26
27    // fase di chiusura
28    // se l'utente ha inserito almeno un voto
29    if (counter != 0) {
30
31        // calcola la media di tutti i voti inseriti
32        double average = (double) total / counter; // evita il troncamento
33
34        // stampa la media con la precisione di due cifre
35        printf("Class average is %.2f\n", average);
36    } // fine di if
37    else { // se non sono stati inseriti voti, stampa un messaggio
38        puts("No grades were entered");
39    } // fine di else
40 } // fine della funzione main

```

```

Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50

```

```

Enter grade, -1 to end: -1
No grades were entered

```

Figura 3.4 Programma per il calcolo della media di una classe con iterazione controllata da sentinella.

Necessità dell'uso delle parentesi graffe nell'istruzione while

Senza le parentesi graffe in questo ciclo while (righe 18 e 25), solo l'istruzione alla riga 19 rimarrebbe all'interno del corpo del ciclo. Il codice verrebbe interpretato in modo scorretto come segue:

```

while (grade != -1)
    total = total + grade; // aggiungi il voto al totale
    counter = counter + 1; // incrementa il contatore

    // ricevi il voto successivo dall'utente
    printf("%s", "Enter grade, -1 to end: "); // prompt per l'input
    scanf("%d", &grade); // leggi il prossimo voto

```

☒ Questo provocherebbe un ciclo infinito se l'utente non inserisse -1 come primo voto.

Conversione esplicita e implicita tra tipi

Le medie spesso sono valori come 7,2 o -93,5 che contengono una parte frazionaria. Questi numeri in virgola mobile possono essere rappresentati dal tipo di dati double. La riga 32 definisce la variabile average come tipo double per catturare il risultato frazionario del nostro calcolo. Normalmente, il risultato del calcolo total / counter (riga 32) è un numero intero poiché total e counter sono entrambe variabili intere. La divisione di due numeri interi produce una **divisione intera** in cui la parte frazionale del calcolo è **troncata** (cioè viene persa). Per realizzare un calcolo in virgola mobile con valori interi dobbiamo prima creare valori temporanei che sono numeri in virgola mobile. Per fare ciò, il C mette a disposizione l'**operatore cast** unario. La riga 32

```
double average = (double) total / counter;
```

include l'operatore cast (double) per creare una copia *temporanea* in virgola mobile del suo operando, total. Il valore memorizzato in total è ancora un numero intero. Un simile uso dell'operatore cast è chiamato **conversione esplicita**. Il calcolo consiste adesso in un valore in virgola mobile (la versione temporanea double di total) diviso per il valore int memorizzato in counter.

Il C richiede che i tipi di dati degli operandi nelle espressioni aritmetiche siano identici. Nelle espressioni con tipi di dati diversi, il compilatore esegue su alcuni operandi un'operazione chiamata **conversione implicita** per assicurarsi che essi siano dello stesso tipo. Per esempio, in un'espressione contenente i tipi di dati int e double vengono create copie degli operandi int e convertite implicitamente nel tipo double. Dopo che total viene convertito esplicitamente in un tipo double, il compilatore crea implicitamente una copia di tipo double di counter, poi esegue una divisione in virgola mobile e ne assegna il risultato in virgola mobile ad average. Esamineremo nel Capitolo 5 le regole del C per la conversione di operandi di tipi differenti.

Gli operatori cast vengono formati ponendo tra parentesi il nome di un tipo. Ogni operatore cast è un **operatore unario** che si applica a un solo operando. Il C supporta anche versioni unarie degli operatori più (+) e meno (-), così potete scrivere espressioni come -7 o $+5$. Gli operatori cast sono associativi da destra a sinistra e hanno la stessa precedenza di altri operatori unari come $+ \text{ unario}$ e $- \text{ unario}$. Questa precedenza è di un livello più alta di quella degli operatori moltiplicativi $*$, $/$ e $\%$.

Formattare numeri in virgola mobile

La Figura 3.4 usa la specifica di conversione $%.2f$ di `printf` (riga 35) per formattare il valore di `average`. La f specifica che sarà stampato un valore in virgola mobile. Il $.2$ è la **precisione**: il valore avrà due (2) cifre alla destra del punto decimale. Se è usato la specifica di conversione $%f$ senza specificare la precisione, viene usata la **precisione predefinita** di 6 cifre alla destra del punto decimale, esattamente come se fosse stata usata la specifica di conversione $%.6f$. Quando i valori in virgola mobile sono stampati con una certa precisione, il valore stampato è **arrotondato** al numero indicato di cifre decimali. Il valore in memoria rimane inalterato. Le seguenti istruzioni stampano i valori 3.45 e 3.4, rispettivamente:

```
printf("%.2f\n", 3.446); // stampa 3.45
printf("%.1f\n", 3.446); // stampa 3.4
```

Note sui numeri in virgola mobile

Benché non siano sempre “precisi al 100%”, i numeri in virgola mobile hanno numerose applicazioni. Per esempio, quando parliamo di una “normale” temperatura corporea di 98,6 gradi Fahrenheit, non abbiamo bisogno di essere precisi fino a un grande numero di cifre. Quando osserviamo la temperatura su un termometro e la leggiamo come 98,6, essa può in realtà essere 98,5999473210643. Il punto qui è che indicare questo numero semplicemente con 98,6 va benissimo per la maggior parte delle applicazioni. Diremo di più su questo argomento in seguito.

Un altro possibile caso di numeri in virgola mobile è quello che si può presentare con una divisione. Quando dividiamo 10 per 3, il risultato è 3,333333... con la sequenza dei 3 che si ripete all’infinito. Il computer assegna solo una quantità fissata di spazio per contenere un tale valore, così il valore in virgola mobile memorizzato può essere soltanto un’approssimazione. Usare numeri in virgola mobile in un modo che presume che essi siano rappresentati precisamente può portare a risultati scorretti. I numeri in virgola mobile sono rappresentati solo approssimativamente dalla maggior parte dei computer. Per questo motivo non dovreste confrontare i valori in virgola mobile per l’uguaglianza.

✓ Autovalutazione

1. (*Completare*) L’iterazione controllata da sentinella è spesso chiamata iterazione _____ perché il numero di iterazioni non è noto prima che il ciclo inizi l’esecuzione.

Risposta: indefinita.

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) Molti programmi possono essere divisi logicamente in tre fasi: una fase di inizializzazione che inizializza le variabili del programma, una fase di elaborazione che riceve in ingresso i valori dei dati e regola conseguentemente le variabili del programma, e una fase di chiusura che calcola e stampa i risultati finali.
- b) Il processo di affinamento graduale top-down termina quando l’algoritmo in pseudocodice è specificato con sufficiente dettaglio tale da poter convertire lo pseudocodice direttamente in C.
- c) L’esperienza dimostra che, quando si deve risolvere un problema usando un computer, la parte più difficile consiste nel produrre un programma funzionante in C a partire dall’algoritmo.
- d) Molti programmatore scrivono direttamente i programmi senza avvalersi di strumenti per il loro sviluppo come lo pseudocodice. Ritengono che l’obiettivo finale sia quello di risolvere il problema su un computer e che la scrittura dello pseudocodice sia soltanto una perdita di tempo nella realizzazione del programma.

Risposta: c) è falsa. L'esperienza dimostra che, quando si deve risolvere un problema su un computer, la parte più difficile consiste nello sviluppare l'algoritmo per la soluzione. Una volta che è stato definito il corretto algoritmo, in genere è poi semplice produrre un programma funzionante in C.

3. (*Segmento di programma*) Quando i valori in virgola mobile sono stampati con precisione, il valore stampato è arrotondato. Scrivete istruzioni che stampino il valore 98.5999473210643 rispettivamente con una, quattro e dieci cifre di precisione, e specificate cosa viene visualizzato in ciascun caso.

Risposta:

```
printf("%.1f\n", 98.5999473210643); // stampa 98.6
printf("%.4f\n", 98.5999473210643); // stampa 98.5999
printf("%.10f\n", 98.5999473210643); // stampa 98.5999473211
```

3.10 Formulare algoritmi con affinamento graduale top-down, caso pratico 3: istruzioni di controllo annidate

Risolviamo un altro problema completo. Formuleremo ancora una volta l'algoritmo usando lo pseudocodice e l'affinamento graduale top-down, e scriveremo un corrispondente programma in C. Abbiamo visto che le istruzioni di controllo possono essere accatastate una dopo l'altra (in sequenza) proprio come fa un bambino con i blocchi di costruzioni. In questo caso pratico vedremo l'unico altro modo strutturato in cui le istruzioni di controllo possono essere connesse tra loro in C, ossia attraverso l'**annidamento** di un'istruzione di controllo all'interno di un'altra. Considerate il seguente problema:

Un college offre un corso che prepara gli studenti all'esame di stato di licenza per intermediari immobiliari. Lo scorso anno 10 degli studenti che hanno completato questo corso hanno fatto l'esame di licenza. Naturalmente il college vuole sapere come gli studenti si sono classificati all'esame. Vi si chiede di scrivere un programma per riassumere i risultati. Vi viene dato un elenco di questi 10 studenti. Accanto a ogni nome viene scritto un 1 se lo studente ha superato l'esame o un 2 se ha fallito.

Il vostro programma deve analizzare i risultati dell'esame come segue:

1. Leggere ogni risultato dell'esame (cioè un 1 o un 2). Stampare il messaggio di prompt "Enter result" ogni volta che il programma richiede un altro risultato dell'esame.
2. Contare il numero dei risultati dell'esame di ogni tipo.
3. Mostrare un riepilogo dei risultati dell'esame indicando il numero degli studenti che lo hanno superato e il numero di quelli che sono stati respinti.
4. Se più di otto studenti hanno superato l'esame, stampare il messaggio "Bonus to instructor!".

Dopo aver letto attentamente il testo del problema, facciamo le seguenti osservazioni:

1. Il programma deve elaborare 10 risultati di un esame. Sarà usato un ciclo controllato da contatore.
2. Ogni risultato dell'esame è un numero, un 1 o un 2. Ogni volta che legge un risultato, il programma deve stabilire se il numero è un 1 o un 2. Nel nostro algoritmo controlliamo se si tratta di un 1. Se il numero non è un 1, presumiamo che sia un 2. L'Esercizio 3.27 vi chiede di garantire che ogni risultato dell'esame sia un 1 o un 2.
3. Sono usati due contatori, uno per contare il numero di studenti che hanno superato l'esame e uno per contare il numero di studenti che sono stati respinti.
4. Dopo che il programma ha elaborato tutti i risultati, deve decidere se più di otto studenti hanno superato l'esame e, in caso affermativo, stampare "Bonus to Instructor!".

Rappresentazione al livello top in pseudocodice

Procediamo con l'affinamento graduale top-down. Iniziamo con una rappresentazione al livello top in pseudocodice:

Analizza i risultati dell'esame e decidi se l'istruttore deve ricevere un bonus

Ancora una volta, è importante mettere in rilievo che il livello top è una rappresentazione completa del programma, ma è probabile che siano necessari diversi affinamenti prima che lo pseudocodice possa essere naturalmente tradotto in un programma in C.

Primo affinamento

Il nostro primo affinamento è:

- Inizializza le variabili
- Leggi i dieci voti dell'esame e conta le promozioni e le bocciature
- Stampa un riepilogo dei risultati dell'esame e decidi se l'istruttore deve ricevere un bonus

Anche qui, pur avendo una rappresentazione *completa* dell'intero programma, è necessario un ulteriore affinamento.

Secondo affinamento

Ora individuiamo le variabili specifiche. Sono necessari dei contatori per registrare le promozioni e le bocciature, un contatore per controllare il processo di iterazione e una variabile per memorizzare l'input dell'utente. L'istruzione in pseudocodice

Inizializza le variabili
si può affinare come segue:

- Inizializza il contatore delle promozioni a zero
- Inizializza il contatore delle bocciature a zero
- Inizializza il contatore del numero di studenti a uno

Vengono inizializzati solo il contatore e i totali. L'istruzione in pseudocodice

richiede un ciclo che in successione legga il risultato di ogni esame. Qui si sa in anticipo che vi sono precisamente dieci risultati dell'esame, così l'iterazione controllata da contatore è quella appropriata. Dentro il ciclo (ossia **annidata** all'interno del ciclo) un'istruzione di selezione doppia determinerà se il risultato di ogni esame è una promozione o una bocciatura e incrementerà di conseguenza il contatore corrispondente. L'affinamento della precedente istruzione in pseudocodice è allora

- Finché il contatore del numero di studenti è minore o uguale a dieci
 - Leggi il risultato successivo dell'esame
 - Se lo studente ha superato l'esame
 - Aggiungi uno al contatore delle promozioni
 - altrimenti
 - Aggiungi uno al contatore delle bocciature
 - Aggiungi uno al contatore del numero di studenti

L'istruzione in pseudocodice

Stampa un riepilogo dei risultati dell'esame e decidi se l'istruttore deve ricevere un bonus
può essere affinata come segue:

- Stampa il numero di promozioni
- Stampa il numero di bocciature
- Se più di otto studenti sono stati promossi
 - Stampa "Bonus to instructor!"

Secondo affinamento completo

La Figura 3.5 contiene il secondo affinamento completo. Usiamo righe vuote per migliorare la leggibilità del programma.

```

1 Inizializza il contatore delle promozioni a zero
1 Inizializza il contatore delle bocciature a zero
1 Inizializza il contatore del numero di studenti a uno
1
1 Finché il contatore del numero di studenti è minore o uguale a dieci
1   Leggi il risultato successivo dell'esame
1
1   Se lo studente ha superato l'esame
1     Aggiungi uno al contatore delle promozioni
1   altrimenti
1     Aggiungi uno al contatore delle bocciature
1
1   Aggiungi uno al contatore del numero di studenti
1
1 Stampa il numero di promozioni
1 Stampa il numero di bocciature
1 Se più di otto studenti sono stati promossi
2   Stampa "Bonus to instructor!"
```

Figura 3.5 Pseudocodice per il problema dei risultati dell'esame.

Implementare l'algoritmo

Questo pseudocodice è ora sufficientemente affinato per la conversione in C. Nella Figura 3.6 sono mostrati il programma in C e due esempi di esecuzione.

```

1 // fig03_06.c
2 // Analisi dei risultati dell'esame.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     // inizializza le variabili nelle definizioni
8     int passes = 0;
9     int failures = 0;
10    int student = 1;
11
12    // processa 10 studenti usando un ciclo controllato da contatore
13    while (student <= 10) {
14        // richiedi all'utente un valore in ingresso
15        printf("%s", "Enter result (1=pass,2=fail): ");
16        int result = 0; // risultato di un esame
17        scanf("%d", &result);
18
19        // se il risultato e' 1, incrementa il numero di promozioni
20        if (result == 1) {
21            passes = passes + 1;
22        } // fine di if
23        else { // altrimenti, incrementa il numero di bocciature
24            failures = failures + 1;
25        } // fine di else
```

```

26
27     student = student + 1; // incrementa il contatore degli studenti
28 } // fine di while
29
30 // fase di terminazione; stampa i risultati
31 printf("Passed %d\n", passes);
32 printf("Failed %d\n", failures);
33
34 // se sono stati promossi piu' di 8 studenti, stampa "Bonus to instructor!"
35 if (passes > 8) {
36     puts("Bonus to instructor!");
37 } // fine di if
38 } // fine della funzione main

```

```

Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 2
Enter Result (1=pass, 2=fail): 2
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 2
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 2
Passed 6
Failed 4

```

```

Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 2
Enter Result (1=pass, 2=fail): 1
Passed 9
Failed 1
Bonus to instructor!

```

Figura 3.6 Analisi dei risultati dell'esame.

✓ Autovalutazione

1. (*Completere*) Le istruzioni di controllo possono essere “accatastate” una dopo l’altra (in sequenza) proprio come fa un bambino con i blocchi di costruzioni. L’unico altro modo strutturato in cui le istruzioni di controllo possono essere connesse tra loro in C è l’ _____, cioè il posizionamento di un’istruzione di controllo all’interno di un’altra.

Risposta: annidamento.

3.11 Operatori di assegnazione

Il C fornisce diversi operatori di assegnazione per abbreviare le espressioni di assegnazione. Per esempio, l'istruzione

`c = c + 3;`

può essere abbreviata con l'**operatore di assegnazione per l'addizione** `+=` come in

`c += 3;`

L'operatore `+=` aggiunge il valore dell'espressione sulla destra dell'operatore al valore della variabile sulla sinistra dell'operatore e memorizza il risultato in tale variabile. Così, l'assegnazione `c += 3` aggiunge 3 al valore corrente di `c`. La tabella seguente mostra gli operatori di assegnazione per l'aritmetica, alcune espressioni di esempio che usano questi operatori e la loro spiegazione:

Operatore di assegnazione	Esempio di espressione	Spiegazione	Assegna
<i>Panete: int c = 3, d = 5, e = 4, f = 6, g = 12;</i>			
<code>+=</code>	<code>c += 7</code>	<code>c = c + 7</code>	10 a c
<code>-=</code>	<code>d -= 4</code>	<code>d = d - 4</code>	1 a d
<code>*=</code>	<code>e *= 5</code>	<code>e = e * 5</code>	20 a e
<code>/=</code>	<code>f /= 3</code>	<code>f = f / 3</code>	2 a f
<code>%=</code>	<code>g %= 9</code>	<code>g = g % 9</code>	3 a g

✓ Autovalutazione

1. (*Completare*) L'istruzione

`b = b * 5;`

può essere abbreviata con l'operatore di assegnazione per la moltiplicazione `*=` come _____.

Risposta: `b *= 5;`

2. (*Completare*) Cosa fa l'assegnazione `c -= 3;?` _____

Risposta: Sottrae 3 dal valore corrente di `c`.

3.12 Operatori di incremento e di decremento

L'**operatore di incremento unario** `(++)` aggiunge uno a una variabile intera, mentre l'**operatore di decremento unario** `(--)` sottrae uno da essa. La seguente tabella riepiloga le due versioni di ciascun operatore:

Operatore	Esempio di espressione	Spiegazione
<code>++</code>	<code>++a</code>	Incrementa <code>a</code> di 1, quindi usa il nuovo valore di <code>a</code> nell'espressione in cui si trova <code>a</code> .
<code>++</code>	<code>a++</code>	Usa il valore corrente di <code>a</code> nell'espressione in cui si trova <code>a</code> , quindi incrementa <code>a</code> di 1.
<code>--</code>	<code>--b</code>	Decrementa <code>b</code> di 1, quindi usa il nuovo valore di <code>b</code> nell'espressione in cui si trova <code>b</code> .
<code>--</code>	<code>b--</code>	Usa il valore corrente di <code>b</code> in cui si trova <code>b</code> , quindi decrementa <code>b</code> di 1.

Se una variabile va incrementata di 1, si può usare l'operatore di incremento `++` piuttosto che le espressioni `c = c + 1` o `c += 1`. Se gli operatori `++` o `--` sono posti prima di una variabile (cioè sono prefissi), sono definiti rispettivamente **operatori di preincremento** o **di predecremento**. Se gli operatori `++` o `--` sono posti dopo una variabile (cioè sono postfissi), sono definiti rispettivamente **operatori di postincremento** o **di postdecremento**. Per convenzione, gli operatori unari dovrebbero essere posizionati accanto ai loro operandi senza spazi intermedi.

La Figura 3.7 illustra la differenza tra le versioni di preincremento e di postincremento dell'operatore `++`. Postincrementare la variabile `c` fa sì che essa sia incrementata *dopo* essere stata usata nell'istruzione `printf`. Preincrementare la variabile `c` fa sì che essa sia incrementata *prima* di essere usata nell'istruzione `printf`. Il programma mostra il valore di `c` prima e dopo l'uso dell'operatore `++`. L'operatore di decremento (`--`) funziona in modo simile.

```

1 // fig03_07.c
2 // Preincremento e postincremento.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7     // illustrazione del postincremento
8     int c = 5; // assegna 5 a c
9     printf("%d\n", c); // stampa 5
10    printf("%d\n", c++); // stampa 5 e poi postincrementa
11    printf("%d\n\n", c); // stampa 6
12
13    // illustrazione del preincremento
14    c = 5; // assegna 5 a c
15    printf("%d\n", c); // stampa 5
16    printf("%d\n", ++c); // preincrementa e poi stampa 6
17    printf("%d\n", c); // stampa 6
18 } // fine della funzione main

```

```

5
5
6

5
6
6

```

Figura 3.7 Preincremento e postincremento.

Le tre istruzioni di assegnazione nella Figura 3.6

```

passes = passes + 1;
failures = failures + 1;
student = student + 1;

```

si possono scrivere più concisamente con gli operatori di assegnazione come

```

passes += 1;
failures += 1;
student += 1;

```

con operatori di preincremento come

```
++passes;
++failures;
++student;
```

o con operatori di postincremento come

```
passes++;
failures++;
student++;
```

Quando si incrementa o si decrementa una variabile in un'istruzione in cui compare solo la stessa variabile, le forme di preincremento e postincremento hanno lo stesso effetto. È solo quando una variabile appare nel contesto di un'espressione più ampia che il preincremento e il postincremento hanno effetti differenti (e lo stesso vale per il predecremento e il postdecremento).

Soltanto il semplice nome di una variabile può essere usato come operando di un operatore `++` o `--`. Tentare di usare l'operatore di incremento o di decremento su un'espressione diversa dal semplice nome di una variabile è un errore di sintassi: per esempio, scrivere `++(x + 1)`.

Il C generalmente non specifica l'ordine in cui saranno calcolati gli operandi di un operatore, anche se vedremo qualche eccezione a ciò per alcuni operatori nel prossimo capitolo. Per evitare errori subdoli, dovete usare gli operatori `++` e `--` unicamente in istruzioni nelle quali viene modificata una sola variabile.

La tabella seguente elenca in ordine decrescente di precedenza gli operatori introdotti fino a questo punto.

Operatori	Associatività	Tipo
<code>++(postfisso) --(postfisso)</code>	da destra a sinistra	postfisso
<code>+ - (tipo) ++(prefisso) --(prefisso)</code>	da destra a sinistra	unario
<code>* / %</code>	da sinistra a destra	moltipicativo
<code>+ -</code>	da sinistra a destra	additivo
<code>< <= > >=</code>	da sinistra a destra	relazionale
<code>== !=</code>	da sinistra a destra	di uguaglianza
<code>? :</code>	da destra a sinistra	condizionale
<code>= += -= *= /= %=</code>	da destra a sinistra	di assegnazione

La terza colonna denomina i vari gruppi di operatori. Notate che l'operatore condizionale (`? :`), gli operatori unari di incremento (`++`), decremento (`--`), più (`+`), meno (`-`), cast e gli operatori di assegnazione `=`, `+=`, `-=`, `*=`, `/=` e `%=` sono associativi da destra a sinistra. Gli altri operatori sono associativi da sinistra a destra.

✓ Autovalutazione

1. (Scelta multipla) Dato il codice:

```
--i;
```

quale delle seguenti affermazioni descrive cosa fa questo codice?

- a) Incrementa `i` di 1, quindi usa il nuovo valore di `i` nell'espressione in cui si trova `i`.
- b) Usa il valore corrente di `i` nell'espressione in cui si trova `i`, quindi incrementa `i` di 1.
- c) Decrementa `i` di 1, quindi usa il nuovo valore di `i` nell'espressione in cui si trova `i`.
- d) Usa il valore corrente di `i` nell'espressione in cui si trova `i`, quindi decrementa `i` di 1.

Risposta: c).

2. (*Cosa fa questo codice?*) Cosa stampa questo programma come valore finale di `x`?

```

1 #include <stdio.h>
2
3 int main(void) {
4     int x = 7;
5     printf("%d\n", x);
6     printf("%d\n", x++);
7     printf("%d\n\n", x);
8     x = 8;
9     printf("%d\n", x);
10    printf("%d\n", ++x);
11    printf("%d\n", x);
12 }
```

Risposta: 9.

3. (*Scelta multipla*) Quale delle seguenti espressioni contiene un errore di sintassi?

- a) `++x + 1`
- b) `x++ + x`
- c) `++(x) + 1`
- d) `++(x + 1)`

Risposta: d). Soltanto il semplice nome di una variabile può essere usato come operando di un operatore `++` o `--`. Tentare di usare l'operatore di incremento o di decremento su un'espressione diversa dal semplice nome di una variabile è un errore di sintassi: per esempio, scrivere `++(x + 1)`.



3.13 Programmazione sicura in C

Overflow aritmetico

La Figura 2.4 presentava un programma di addizione che calcolava la somma di due valori `int` con l'istruzione

```
sum = integer1 + integer2; // assegna il totale a sum
```



Persino questa semplice istruzione ha un problema potenziale: addizionare numeri interi potrebbe produrre un valore troppo grande da memorizzare in una variabile `sum` di tipo `int`. Ciò è noto come **overflow aritmetico** e può causare un comportamento indefinito, con la possibilità che il sistema venga esposto ad attacchi.

Le costanti `INT_MAX` e `INT_MIN` rappresentano i valori massimi e minimi specifici della piattaforma che si possono memorizzare in una variabile `int`. Queste costanti sono definite nel file di intestazione `<limits.h>`. Vi sono costanti simili per gli altri tipi interi che introdurremo nel prossimo capitolo. Potete vedere i valori relativi alla vostra piattaforma per queste costanti aprendo il file di intestazione `<limits.h>` in un editor di testo.²

Viene considerata una buona pratica assicurarsi che, prima di eseguire calcoli aritmetici come quello sopra, questi non daranno luogo a overflow. Per un esempio, consultate il sito del CERT

<https://wiki.sei.cmu.edu/confluence/display/c/>

Cercate la linea guida INT32-C. Il codice utilizza gli operatori `&&` (AND logico) e `||` (OR logico), che analizzeremo nel Capitolo 4. Nel codice sviluppato a livello industriale dovreste eseguire controlli come questi per tutti i calcoli. Nei capitoli successivi mostreremo altre tecniche di programmazione per trattare questi errori.

`scanf_s` e `printf_s`

L'Annex K dello standard C11 ha introdotto versioni più sicure di `printf` e `scanf` chiamate `printf_s` e `scanf_s`. Analizzeremo queste funzioni e le questioni di sicurezza corrispondenti nei Paragrafi 6.13 e 7.13. L'Annex K è designato come opzionale, per cui non sarà implementato da tutti i fornitori del linguaggio C.

2. Usate la funzione di ricerca del vostro sistema per trovare il file `limits.h`.

In particolare, i compilatori GNU C++ e Clang C++ non implementano l'Annex K, quindi usare le funzioni `scanf_s` e `printf_s` potrebbe compromettere la portabilità del vostro codice su diversi compilatori.

Microsoft ha implementato le proprie versioni in Visual C++ di `printf_s` e `scanf_s` prima della pubblicazione dello standard C11, e il suo compilatore ha cominciato immediatamente a far emettere dei warning (avvertimenti) per ogni chiamata di `scanf`. I warning dicono che `scanf` è deprecata – non deve essere più utilizzata – e che voi dovreste prendere in considerazione invece l'uso di `scanf_s`. Microsoft ora considera errori quelli che erano warning relativi a `scanf`. Un programma con funzioni `scanf` non verrà compilato su Visual C++ e non potrà essere eseguito.

Molte organizzazioni hanno standard di codifica che richiedono che il codice venga compilato senza messaggi di warning. Vi sono due modi per eliminare i warning per `scanf` in Visual C++: potete usare `scanf_s` invece di `scanf` oppure potete disattivare questi warning. Riguardo alle istruzioni di input che abbiamo usato finora, gli utenti di Visual C++ possono semplicemente sostituire `scanf` con `scanf_s`. Potete disattivare i messaggi di warning in Visual C++ come segue:

1. Digitate ***Alt F7*** per far apparire la finestra **Property Pages** per il vostro progetto.
2. Nella colonna sinistra, espandete **Configuration Properties > C/C++ > Preprocessor**.
3. Nella colonna destra, alla fine del valore per **Preprocessor Definitions**, inserite

`:_CRT_SECURE_NO_WARNINGS`

4. Fate clic su **OK** per salvare le modifiche.

Non riceverete più warning per `scanf` (o per qualunque altra funzione che Microsoft ha deprecato per ragioni simili). Per codice a livello industriale è sconsigliato disattivare i warning. Diremo di più su come usare `scanf_s` e `printf_s` in un successivo paragrafo sulle linee guida per il codice C sicuro.

✓ Autovalutazione

1. (*Completare*) Addizionare due interi che producono come risultato un valore troppo grande da memorizzare in una variabile `int` è noto come _____ aritmetico e può causare un comportamento indefinito, con la possibilità che il sistema venga esposto ad attacchi.

Risposta: overflow.

2. (*Completare*) I valori massimi e minimi specifici della piattaforma che si possono memorizzare in una variabile `int` sono rappresentati rispettivamente dalle costanti `INT_MAX` e `INT_MIN`, che sono definite nel file di intestazione _____.

Risposta: `<limits.h>`.

3.14 Riepilogo

Paragrafo 3.1 Introduzione

- Prima di scrivere un programma per risolvere un problema particolare, dovete avere una comprensione completa del problema e un approccio pianificato con cura per risolverlo.

Paragrafo 3.2 Algoritmi

- La soluzione per qualsiasi problema di calcolo implica l'esecuzione di una serie di **azioni** in un **ordine** specifico.
- Un **algoritmo** è una **procedura** per risolvere un problema che consta in un insieme di azioni da eseguire e in un dato ordine in cui esse vanno eseguite.

Paragrafo 3.3 Pseudocodice

- Lo **pseudocodice** è un linguaggio artificiale e informale che aiuta a sviluppare algoritmi.

- Lo pseudocodice è simile al linguaggio di ogni giorno; non è un vero e proprio linguaggio di programmazione per computer.
- I programmi in pseudocodice aiutano a “riflettere bene” su un programma.
- Lo pseudocodice consiste puramente di caratteri; potete scriverlo usando un editor di testo.
- I programmi in pseudocodice preparati con cura si possono convertire facilmente in corrispondenti programmi in C.
- Lo pseudocodice è costituito solamente da azioni e decisioni.

Paragrafo 3.4 Strutture di controllo

- Normalmente, le istruzioni in un programma vengono eseguite una dopo l'altra nell'ordine in cui vengono scritte. Questa è chiamata **esecuzione sequenziale**.
- Varie istruzioni in C permettono di specificare che l'istruzione successiva da eseguire può essere diversa dalla successiva in sequenza. Ciò è chiamato **trasferimento del controllo**.
- La **programmazione strutturata** è diventata quasi sinonimo di “**eliminazione del goto**”.
- I programmi strutturati sono più chiari e più facili da correggere e modificare, oltre ad avere più probabilità di essere esenti da errori.
- Tutti i programmi possono essere scritti in termini di **strutture di controllo sequenziale, di selezione e di iterazione**.
- A meno che non sia istruito per agire diversamente, il computer esegue automaticamente le istruzioni in C in sequenza.
- Un **diagramma di flusso** è la rappresentazione grafica di un algoritmo disegnato usando **rettangoli, rombi, rettangoli arrotondati e cerchietti**, collegati da frecce chiamate **linee di flusso**.
- Il **simbolo rettangolo (azione)** indica qualsiasi tipo di azione, come un calcolo o un'operazione di input/output.
- Le **linee di flusso** indicano l'ordine in cui sono eseguite le azioni.
- Quando si disegna un diagramma di flusso che rappresenta un algoritmo completo, il primo simbolo usato è un rettangolo arrotondato contenente la parola “Inizio”, mentre l'ultimo simbolo usato è un rettangolo arrotondato contenente la parola “Fine”. Quando si disegna soltanto una porzione di algoritmo, omettiamo i simboli rettangolo arrotondato e al loro posto usiamo dei simboli cerchietto chiamati **simboli connettori**.
- L'**istruzione di selezione singola if** seleziona o ignora una singola azione (o un gruppo di azioni).
- L'**istruzione di selezione doppia if...else** seleziona tra due azioni (o gruppi di azioni).
- L'**istruzione di selezione multipla switch** seleziona tra più azioni differenti in base al valore di un'espressione.
- Il C fornisce tre tipi di **istruzioni di iterazione** (chiamate anche istruzioni di ripetizione), ossia **while, do...while e for**.
- I segmenti di diagramma di flusso relativi alle istruzioni di controllo possono essere **accatastati**, ovvero attaccati l'uno all'altro collegando il punto di uscita di un'istruzione di controllo al punto di entrata della successiva.
- Le istruzioni di controllo possono essere **annidate**.
- Il C usa **istruzioni di controllo a un solo ingresso/una sola uscita**.

Paragrafo 3.5 Istruzione di selezione if

- Le strutture di selezione sono usate per scegliere fra linee alternative di azioni.
- Il **simbolo di decisione (rombo)** indica che deve essere presa una decisione.
- L'espressione di un **simbolo di decisione** solitamente è una condizione che può essere *vera* o *falsa*. Il simbolo di decisione ha due linee di flusso che emergono da esso e indicano le direzioni da prendere quando l'espressione è *vera* o *falsa*.

- Una decisione può basarsi su qualsiasi espressione: se l'espressione assume il valore zero è trattata come *falsa* e se assume un valore diverso da zero è trattata come *vera*.

Paragrafo 3.6 Istruzione di selezione if...else

- L'operatore condizionale (`:?`) è strettamente correlato all'istruzione if...else.
- L'operatore condizionale è il solo operatore ternario del C: esso si applica a tre operandi. Il primo operando è una condizione; il secondo operando è il valore dell'espressione condizionale se la condizione è *vera*; il terzo operando è il valore dell'espressione condizionale se la condizione è *falsa*.
- Le istruzioni annidate if...else effettuano test per più casi ponendo istruzioni if...else dentro altre istruzioni if...else.
- Un insieme di istruzioni contenute all'interno di una coppia di parentesi graffe è chiamato **istruzione composta o blocco**.
- Un errore di sintassi viene individuato dal compilatore. Un errore logico ha il suo effetto al momento dell'esecuzione. Un errore logico irreversibile fa sì che il programma fallisca e termini prematuramente. Un errore logico non irreversibile permette a un programma di continuare l'esecuzione ma con la produzione di risultati scorretti.

Paragrafo 3.7 Istruzione di iterazione while

- L'istruzione di iterazione while specifica che un'azione va ripetuta finché una condizione è *vera*. Alla fine la condizione diventerà *falsa*. A questo punto l'iterazione termina e viene eseguita la prima istruzione dopo l'istruzione di iterazione.

Paragrafo 3.8 Formulare algoritmi, caso pratico 1: iterazione controllata da contatore

- L'iterazione controllata da contatore utilizza una variabile chiamata **contatore** per specificare il numero di volte in cui deve essere eseguito un insieme di istruzioni.
- L'iterazione controllata da contatore è spesso chiamata **iterazione definita** perché il numero di iterazioni è noto prima che il ciclo inizi l'esecuzione.
- Un **totale** è una variabile usata per accumulare la somma di una serie di valori. Le variabili usate per memorizzare i totali devono normalmente essere inizializzate a zero.
- Un **contatore** è una variabile usata per contare. Le variabili contatore sono normalmente inizializzate a zero o a uno, a seconda del loro uso.
- Una **variabile non inizializzata** contiene un valore "spazzatura", l'ultimo valore memorizzato nella locazione di memoria riservata per quella variabile.

Paragrafo 3.9 Formulare algoritmi con affinamento graduale top-down, caso pratico 2: iterazione controllata da sentinella

- Un **valore sentinella** (chiamato anche **valore di segnale**, **valore fittizio** o **valore di flag**) è usato in un ciclo controllato da sentinella per indicare la "fine dell'inserimento dei dati".
- L'iterazione controllata da sentinella è spesso chiamata **iterazione indefinita** perché il numero di iterazioni non è noto prima che il ciclo inizi l'esecuzione.
- Il valore sentinella deve essere scelto in modo tale che non possa essere confuso con un valore di input accettabile.
- Nell'affinamento graduale top-down, il livello **top** è un'istruzione che indica la funzione complessiva realizzata dal programma. È una rappresentazione completa di un programma. In un **processo di affinamento** suddividiamo il livello top in compiti più piccoli e li elenchiamo in ordine di esecuzione.
- Il tipo **double** rappresenta numeri con punto decimale (chiamati **numeri in virgola mobile**).
- Quando si dividono due numeri interi, la parte frazionale del risultato viene **troncata**.

- Per compiere un calcolo in virgola mobile con valori interi, dovete trasformare i numeri interi in numeri in virgola mobile. Il C fornisce l'**operatore cast unario** (`double`) per effettuare questa azione.
- Gli operatori cast eseguono **conversioni esplicite**.
- Il C richiede che gli operandi nelle espressioni aritmetiche abbiano lo stesso tipo di dati. Per assicurare ciò, il compilatore esegue la **conversione implicita** sugli operandi selezionati.
- Un operatore cast è formato mettendo delle parentesi attorno al nome di un tipo. L'operatore cast è un **operatore unario**: ha soltanto un operando.
- Gli **operatori cast sono associativi da destra a sinistra** e hanno la stessa precedenza di altri operatori unari come + unario e - unario. Questa precedenza è di un livello più alta di quella di *, / e %.
- La specifica di conversione %.2f di `printf` indica che il valore in virgola mobile sarà stampato con due cifre alla destra del numero decimale. Se si usa la specifica di conversione %f (senza indicare la precisione), la **precisione predefinita** è pari a 6.
- Quando i valori in virgola mobile sono stampati con una data precisione, il valore viene **arrotondato**, per esigenze di stampa, al numero indicato di cifre decimali.

Paragrafo 3.11 Operatori di assegnazione

- Il C fornisce diversi operatori di assegnazione per **abbreviare le espressioni di assegnazione**.
- L'operatore += aggiunge il valore dell'espressione alla sua destra al valore della variabile alla sua sinistra e memorizza il risultato nella variabile alla sua sinistra.
- Gli operatori di assegnazione sono forniti per ciascuno degli operatori binari +, -, *, / e %.

Paragrafo 3.12 Operatori di incremento e di decremento

- Il C fornisce l'**operatore di incremento unario**, ++, e l'**operatore di decremento unario**, --, da usare con tipi interi.
- Se gli operatori ++ o -- sono posti prima di una variabile, sono detti rispettivamente **operatori di preincremento o di predecremento**. Se gli operatori ++ o -- sono posti dopo una variabile, sono detti rispettivamente **operatori di postincremento o di postdecremento**.
- Preincrementare (predecrementare) una variabile fa sì che essa venga incrementata (decrementata) di 1, dopodiché il nuovo valore della variabile viene usato nell'espressione in cui compare.
- Postincrementare (postdecrementare) una variabile fa sì che venga utilizzato il valore corrente della variabile nell'espressione in cui compare, dopodiché il valore della variabile viene incrementato (decrementato) di 1.
- Quando si incrementa o si decrementa una variabile in un'istruzione in cui compare la singola variabile, le forme di preincremento e postincremento hanno lo stesso effetto. Quando compare una variabile nel contesto di un'espressione più ampia, il preincremento e il postincremento hanno effetti differenti (e lo stesso avviene per il predecremento e il postdecremento).

Paragrafo 3.13 Programmazione sicura in C

- L'addizione di numeri interi può produrre un valore troppo grande da memorizzare in una variabile `int`. Ciò è noto come **overflow aritmetico** e può provocare un comportamento imprevedibile al momento dell'esecuzione, rendendo possibile l'esposizione del sistema ad attacchi.
- I valori massimi e minimi che si possono memorizzare in una variabile `int` sono rappresentati rispettivamente con le costanti `INT_MAX` e `INT_MIN` definite nel file di intestazione `<limits.h>`.
- Viene considerata una buona pratica assicurarsi che i calcoli aritmetici non daranno luogo a overflow prima della loro esecuzione. Nel codice a livello industriale si devono eseguire controlli per tutti i calcoli che possono provocare un overflow o un underflow.
- L'**Annex K** dello standard C11 introduce **versioni più sicure di printf e scanf** chiamate `printf_s` e `scanf_s`. L'Annex K è designato come opzionale, quindi non tutti i fornitori di compilatori C lo implementeranno.

- Microsoft ha implementato le proprie versioni di `printf_s` e `scanf_s` prima della pubblicazione dello standard C11 e immediatamente ha cominciato a far generare dei warning per ogni chiamata di `scanf`. I warning dicono che `scanf` è deprecata (non deve più essere utilizzata) e che invece dovreste prendere in considerazione l'uso di `scanf_s`.
- Molte organizzazioni hanno standard di codifica che richiedono che il codice sia compilato senza messaggi di warning. Vi sono due modi per eliminare i warning per `scanf` in Visual C++. Potete cominciare a usare immediatamente `scanf_s` o disattivare questi messaggi di warning.

Esercizi di autovalutazione

- 3.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
- Una procedura per risolvere un problema in termini di azioni da eseguire e dell'ordine in cui tali azioni vanno eseguite è chiamata _____.
 - Specificare l'ordine di esecuzione delle istruzioni da parte del computer è chiamato _____.
 - Tutti i programmi possono essere scritti in termini dei tre tipi di istruzioni di controllo: _____, _____ e _____.
 - L'istruzione di selezione _____ è usata per eseguire un'azione quando una condizione è *vera* e un'altra azione quando la condizione è *falsa*.
 - Varie istruzioni raggruppate insieme tra parentesi graffe ({ e }) sono chiamate _____.
 - L'istruzione di iterazione _____ specifica che un'istruzione o un gruppo di istruzioni vanno eseguiti ripetutamente finché una qualche condizione rimane *vera*.
 - L'iterazione di un insieme di istruzioni un numero specifico di volte è chiamata iterazione _____.
 - Quando non si sa in anticipo quante volte sarà ripetuto un insieme di istruzioni si può usare un valore _____ per terminare l'iterazione.
- 3.2 Scrivete quattro differenti istruzioni in C che sommino 1 alla variabile intera `x`.
- 3.3 Scrivete singole istruzioni in C per effettuare ciascuna delle seguenti azioni.
- Moltiplicate la variabile `product` per 2 usando l'operatore *=.
 - Moltiplicate la variabile `product` per 2 usando gli operatori = e *.
 - Verificate se il valore della variabile `count` è maggiore di 10. Se lo è, stampate "Count is greater than 10".
 - Calcolate il resto della divisione di `quotient` per `divisor` e assegnate il risultato a `quotient`. Scrivete questa istruzione in due modi differenti.
 - Stampate il valore 123.4567 con due cifre di precisione. Quale valore viene stampato?
 - Stampate il valore del numero in virgola mobile 3.14159 con tre cifre alla destra del numero decimale. Quale valore viene stampato?
- 3.4 Scrivete un'istruzione in C per eseguire ognuno dei seguenti compiti.
- Definite la variabile `x` come tipo `int` e impostatela a 1.
 - Definite la variabile `sum` come tipo `int` e impostatela a 0.
 - Sommate la variabile `x` alla variabile `sum` e assegnate il risultato alla variabile `sum`.
 - Stampate "The sum is: " seguito dal valore della variabile `sum`.
- 3.5 Mettete insieme le istruzioni che avete scritto nell'Esercizio 3.4 in un programma che calcoli la somma dei numeri interi da 1 a 10. Usate l'istruzione `while` per iterare i calcoli e le istruzioni di incremento. Il ciclo deve terminare quando il valore di `x` diventa 11.
- 3.6 Scrivete singole istruzioni in C per eseguire ciascuno dei seguenti compiti.
- Leggete il valore della variabile intera `x` con `scanf`. Usate la specifica di conversione %d.
 - Leggete il valore della variabile intera `y` con `scanf`. Usate la specifica di conversione %d.
 - Impostate la variabile intera `i` a 1.
 - Impostate la variabile intera `power` a 1.
 - Moltiplicate la variabile intera `power` per `x` e assegnate il risultato a `power`.
 - Incrementate la variabile `i` di 1.

- g) Testate i per vedere se è minore o uguale a y nella condizione di un'istruzione while.
 h) Stampate la variabile intera power con printf.

3.7 Scrivete un programma in C che usi le istruzioni dell'esercizio precedente per calcolare x elevato alla potenza y. Il programma dovrà avere un'istruzione di controllo dell'iterazione while.

3.8 Identificate e correggete gli errori in ognuna delle seguenti istruzioni.

```
a) while (c <= 5) {
    product *= c;
    ++c;
}
b) scanf("%.4f", &value);
c) if (gender == 1) {
    puts("Woman");
}
else {
    puts("Man");
}
```

3.9 Cos'è sbagliato nella seguente istruzione di iterazione while (supponete che il valore di z sia 100), che deve calcolare la somma dei numeri interi da 100 in giù fino a 1?

```
while (z >= 0) {
    sum += z;
}
```

Risposte agli esercizi di autovalutazione

3.1 a) algoritmo. b) controllo del programma. c) sequenza, selezione, iterazione. d) if...else. e) istruzione composta o blocco. f) while. g) controllata da contatore o definita. h) sentinella.

3.2

```
x = x + 1;
x += 1;
++x;
x++;
```

3.3

```
a) product *= 2;
b) product = product * 2;
c) if (count > 10) {
    puts("Count is greater than 10.");
}
d) quotient %= divisor;
quotient = quotient % divisor;
e) printf("%.2f", 123.4567);
viene stampato 123.46.
f) printf("%.3f\n", 3.14159);
viene stampato 3.142.
```

3.4

```
a) int x = 1;
b) int sum = 0;
c) sum += x; o sum = sum + x;
d) printf("The sum is: %d\n", sum);
```

3.5 Si veda sotto.

```
1 // Calcola la somma dei numeri interi da 1 a 10
2 #include <stdio.h>
3
```

```

4 int main(void) {
5     int x = 1; // imposta x
6     int sum = 0; // imposta la somma
7
8     while (x <= 10) { // itera finche' x e' minore o uguale a 10
9         sum += x; // aggiungi x a sum
10        ++x; // incrementa x
11    } // fine di while
12
13    printf("The sum is: %d\n", sum); // stampa la somma
14 } //fine della funzione main

```

- 3.6 a) `scanf("%d", &x);`
 b) `scanf("%d", &y);`
 c) `i = 1;`
 d) `power = 1;`
 e) `power *= x;`
 f) `++i;`
 g) `while (i <= y)`
 h) `printf("%d", power);`

3.7 Si veda sotto.

```

1 // eleva x alla potenza y
2 #include <stdio.h>
3
4 int main(void) {
5     printf("%s", "Enter first integer: ");
6     int x = 0;
7     scanf("%d", &x); // leggi il valore di x
8     printf("%s", "Enter second integer: ");
9     int y = 0;
10    scanf("%d", &y); // leggi il valore di y
11
12    int i = 1;
13    int power = 1; // imposta power
14
15    while (i <= y) { // ripeti finche' i e' minore o uguale a y
16        power *= x; // moltiplica power per x
17        ++i; // incrementa i
18    } // fine di while
19    printf("%d\n", power); // stampa power
20 } // fine della funzione main

```

- 3.8 a) Errore: mancanza della parentesi graffa destra di chiusura nel corpo del while.
 Correzione: aggiungete la parentesi graffa destra di chiusura dopo l'istruzione `++c;`.
 b) Errore: la precisione usata nella specifica di conversione in `scanf`.
 Correzione: rimuovete `.4` dalla specifica di conversione.
 c) Errore: il punto e virgola dopo l'`else` dell'istruzione `if...else` causa un errore logico; il secondo `puts` sarà sempre eseguito.
 Correzione: rimuovete il punto e virgola dopo `else`.

3.9 Il valore della variabile `z` non viene mai cambiato nell'istruzione `while`. Pertanto si genera un ciclo infinito. Per evitare il ciclo infinito si deve decrementare `z` in modo che alla fine diventi uguale a 0.

Esercizi

3.10 Identificate e correggete gli errori. [Nota: potrebbe esserci più di un errore in ogni porzione di codice.]

```
a) if (age >= 65) {
    puts("Age is greater than or equal to 65");
}
else {
    puts("Age is less than 65");
}

b) int x = 1;
int total;

while (x <= 10) {
    total += x;
    ++x;
}

c) While (x <= 100)
    total += x;
    ++x;
d) while (y > 0) {
    printf("%d\n", y);
    ++y;
}
```

3.11 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- La soluzione a qualsiasi problema implica l'esecuzione di una serie di azioni in un _____ specifico.
- Un sinonimo di procedura è _____.
- Una variabile che accumula la somma di diversi numeri è un _____.
- Un valore speciale usato per indicare la "fine dell'inserimento dei dati" è chiamato valore _____,
- _____ , _____ o _____.
- Un _____ è la rappresentazione grafica di un algoritmo.
- In un diagramma di flusso, l'ordine in cui si devono eseguire i passi è indicato da simboli _____.
- I simboli rettangolo corrispondono ai calcoli che sono normalmente eseguiti da istruzioni di _____ e da operazioni di input/output eseguite normalmente per mezzo di chiamate alle funzioni della Libreria Standard _____ e _____.
- L'espressione scritta dentro il simbolo di decisione è chiamata _____.

3.12 Che cosa stampa il seguente programma?

```
1 #include <stdio.h>
2
3 int main(void) {
4     int x = 1;
5     int total = 0;
6
7     while (x <= 10) {
8         int y = x * x;
9         printf("%d\n", y);
10        total += y;
11        ++x;
12    } // fine di while
13 }
```

```

14     printf("Total is %d\n", total);
15 } // fine di main

```

- 3.13 Scrivete singole istruzioni in pseudocodice che indichino ognuna delle seguenti azioni.
- Stampa il messaggio "Inserire due numeri".
 - Assegna la somma delle variabili *x*, *y* e *z* alla variabile *p*.
 - La seguente condizione va verificata in un'istruzione di selezione *if...else*: il valore corrente della variabile *m* è più di due volte maggiore del valore corrente della variabile *v*.
 - Leggi i valori per le variabili *s*, *r* e *t* inseriti da tastiera.
- 3.14 Formulate un algoritmo in pseudocodice per ognuna delle seguenti azioni.
- Leggi due numeri dalla tastiera, calcola la loro somma e stampa il risultato.
 - Leggi due numeri dalla tastiera, e determina e stampa qual è il maggiore dei due (se uno dei due lo è).
 - Leggi una serie di numeri positivi dalla tastiera, e determina e stampa la loro somma. Supponi che l'utente scriva il valore sentinella -1 per indicare la "fine dell'inserimento dei dati".
- 3.15 Stabilite quali delle seguenti affermazioni sono *vere* e quali *false*. Se un'affermazione è *falsa*, spiegate perché.
- L'esperienza ha dimostrato che la parte più difficile della risoluzione di un problema su un computer è quella di produrre un programma in C funzionante.
 - Un valore sentinella deve essere un valore che non può venire confuso con un valore legittimo dei dati.
 - Le linee di flusso indicano le azioni da eseguire.
 - Le condizioni scritte dentro i simboli di decisione contengono sempre operatori aritmetici (ossia, +, -, *, / e %).
 - Nell'affinamento graduale top-down, ogni affinamento è una rappresentazione completa dell'algoritmo.

Per gli Esercizi 3.16-3.20, eseguite ognuno di questi passi:

- Leggere la descrizione del problema.
- Formulare l'algoritmo usando l'affinamento graduale top-down in pseudocodice.
- Scrivere un programma in C.
- Verificare, correggere ed eseguire il programma in C.

3.16 (*Consumo di carburante*) I guidatori sono interessati al consumo effettuato dalle loro automobili. Un guidatore ha tenuto traccia dei vari pieni di carburante, registrando le miglia percorse e i galloni consumati per ogni pieno. Realizzate un programma che utilizzi *scanf* per richiedere l'inserimento delle miglia percorse e dei galloni consumati per ogni pieno. Il programma deve calcolare e stampare le miglia per gallone percorse per ogni pieno. Dopo aver processato tutte le informazioni di input, il programma deve calcolare e stampare le miglia complessive per gallone percorse per tutti i pieni. Ecco un esempio di input/output:

```

Enter the gallons used (-1 to end): 12.8
Enter the miles driven: 287
The miles/gallon for this tank was 22.421875

Enter the gallons used (-1 to end): 10.3
Enter the miles driven: 200
The miles/gallon for this tank was 19.417475

```

```

Enter the gallons used (-1 to end): 5
Enter the miles driven: 120
The miles/gallon for this tank was 24.000000
Enter the gallons used (-1 to end): -1
The overall average miles/gallon was 21.601423

```

3.17 (Verifica dei limiti di credito) Sviluppate un programma in C per determinare se un cliente di un grande magazzino ha superato il limite di credito sul suo conto di addebito. Per ogni cliente sono a disposizione i seguenti dati:

- Numero del conto
- Saldo all'inizio del mese
- Totale delle voci addebitate sul conto del cliente nel mese
- Totale dei crediti applicati nel mese al conto del cliente
- Limite di credito concesso

Il programma deve usare `scanf` per leggere tali dati, calcolare il nuovo saldo ($= \text{saldo iniziale} + \text{addebiti} - \text{crediti}$) e determinare se il nuovo saldo supera il limite di credito del cliente. Per quei clienti il cui limite di credito è stato superato, il programma deve stampare il numero del conto del cliente, il limite di credito, il nuovo saldo e il messaggio "Limite di credito superato". Ecco un esempio di input/output:

```

Enter account number (-1 to end): 100
Enter beginning balance: 5394.78
Enter total charges: 1000.00
Enter total credits: 500.00
Enter credit limit: 5500.00
Account: 100
Credit limit: 5500.00
Balance: 5894.78
Credit Limit Exceeded.

Enter account number (-1 to end): 200
Enter beginning balance: 1000.00
Enter total charges: 123.45
Enter total credits: 321.00
Enter credit limit: 1500.00

Enter account number (-1 to end): 300
Enter beginning balance: 500.00
Enter total charges: 274.73
Enter total credits: 100.00
Enter credit limit: 800.00

Enter account number (-1 to end): -1

```

3.18 (Calcolo delle commissioni sulle vendite) Una grande compagnia chimica paga il suo personale addetto alle vendite su commissione. Il personale addetto alle vendite riceve \$200 alla settimana più il 9% delle vendite lorde per quella settimana. Per esempio, un addetto alle vendite che vende \$5000 di prodotti chimici in una settimana riceve \$200 più il 9% di \$5000, cioè un totale di \$650. Sviluppate un programma che utilizzi `scanf`

per leggere le vendite lorde di ogni addetto alle vendite nell'ultima settimana e calcoli e stampi i guadagni di quell'addetto. Elaborate i dati di un addetto alla volta. Ecco un esempio di input/output:

```
Enter sales in dollars (-1 to end): 5000.00
Salary is: $650.00

Enter sales in dollars (-1 to end): 1234.56
Salary is: $311.11

Enter sales in dollars (-1 to end): -1
```

3.19 (Calcolo degli interessi) L'interesse semplice su un prestito è calcolato con la formula

$$\text{interest} = \text{principal} * \text{rate} * \text{days} / 365;$$

La precedente formula presume che `rate` sia il tasso di interesse annuale, quindi effettua la divisione per 365 (giorni). Sviluppate un programma che utilizzi `scanf` per leggere i valori `principal`, `rate` e `days` per diversi prestiti, e calcoli e stampi l'interesse semplice per ogni prestito, usando la formula precedente. Ecco un esempio di input/output:

```
Enter loan principal (-1 to end): 1000.00
Enter interest rate: .1
Enter term of the loan in days: 365
The interest charge is $100.00

Enter loan principal (-1 to end): 1000.00
Enter interest rate: .08375
Enter term of the loan in days: 224
The interest charge is $51.40

Enter loan principal (-1 to end): -1
```

3.20 (Calcolo del salario) Sviluppate un programma per calcolare lo stipendio lordo di ciascuno dei diversi impiegati. L'azienda paga quanto previsto all'ora per "l'orario lavorativo normale" per le prime 40 ore di lavoro e paga "una volta e mezza" per tutte le ore di lavoro oltre le 40 ore. Vi vengono dati una lista degli impiegati dell'azienda, il numero di ore in cui l'impiegato ha lavorato l'ultima settimana e la paga oraria di ogni impiegato. Il vostro programma deve usare `scanf` per leggere queste informazioni per ogni impiegato e determinare e stampare lo stipendio lordo. Ecco un esempio di input/output:

```
Enter # of hours worked (-1 to end): 39
Enter hourly rate of the worker ($00.00): 10.00
Salary is $390.00

Enter # of hours worked (-1 to end): 40
Enter hourly rate of the worker ($00.00): 10.00
Salary is $400.00

Enter # of hours worked (-1 to end): 41
Enter hourly rate of the worker ($00.00): 10.00
Salary is $415.00

Enter # of hours worked (-1 to end): -1
```

3.21 (Predecrementare e postdecrementare) Scrivete un programma che dimostri la differenza tra predecrementare e postdecrementare usando l'operatore di decremento `--`.

3.22 (Stampare numeri con un ciclo) Scrivete un programma che utilizzi l'iterazione per stampare i numeri da 1 a 10 l'uno accanto all'altro sulla stessa riga con tre spazi tra di loro.

3.23 (Trovare il numero più grande) Il processo di elaborazione che consiste nel trovare il numero più grande (cioè il maggiore di un insieme di numeri) si usa frequentemente nelle applicazioni informatiche. Per esempio, un programma che determina il vincitore di una gara di vendite riceve in ingresso il numero di unità vendute per ogni persona addetta alle vendite. La persona che vende più unità vince la gara. Scrivete un programma in pseudocodice e poi un programma che utilizzi `scanf` per leggere una serie di 10 numeri non-negativi e determini e stampi il maggiore dei numeri. Il vostro programma deve utilizzare tre variabili:

- counter: un contatore per contare fino a 10 (cioè per tenere il conto di quanti numeri siano stati inseriti e per determinare quando tutti e dieci i numeri sono stati elaborati).
- number: il numero corrente inserito nel programma.
- largest: il numero maggiore trovato fino a quel punto.

3.24 (Tabella di output) Scrivete un programma che usi l'iterazione per stampare la seguente tabella di valori. Usate la sequenza di escape tab, `\t`, nell'istruzione `printf` per separare le colonne con tabulazioni.

N	10^*N	100^*N	1000^*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000
6	60	600	6000
7	70	700	7000
8	80	800	8000
9	90	900	9000
10	100	1000	10000

3.25 (Tabella di output) Scrivete un programma che utilizzi l'iterazione per produrre la seguente tabella di valori:

A	$A+2$	$A+4$	$A+6$
3	5	7	9
6	8	10	12
9	11	13	15
12	14	16	18
15	17	19	21

3.26 (Trovare i due numeri più grandi) Usando un approccio simile a quello dell'Esercizio 3.23, trovate i due valori più grandi tra 10 numeri. Potete inserire ogni numero solo *una volta*.

3.27 (Convalidare l'input dell'utente) Modificate il programma della Figura 3.6 per convalidare i suoi input. Per qualunque input, se il valore inserito è diverso da 1 o da 2, continuate l'iterazione finché l'utente non inserisce un valore corretto.

3.28 Che cosa stampa il seguente programma?

```

1 #include <stdio.h>
2
3 int main(void) {
4     int count = 1; // inizializza count
5
6     while (count <= 10) { // ripeti 10 volte
7         // output della riga di testo
8         puts((count % 2) ? "*****" : "++++++");
9         ++count; // incrementa count
10    } // fine di while
11 } // fine della funzione main

```

3.29 Che cosa stampa il seguente programma?

```

1 #include <stdio.h>
2
3 int main(void) {
4     int row = 10; // inizializza row
5
6     while (row >= 1) { // ripeti finche' row < 1
7         int column = 1; // imposta column a 1 all'inizio dell'iterazione
8
9         while (column <= 10) { // ripeti 10 volte
10            printf("%s", (row % 2) ? "<" : ">"); // output
11            ++column; // incrementa column
12        } // fine del while interno
13
14        --row; // decrementa row
15        puts(""); // inizia una nuova riga di output
16    } // fine del while esterno
17 } // fine della funzione main

```

3.30 (*Problema dell'else sospeso*) Determinate l'output per ognuna delle seguenti istruzioni quando x è 9 e y è 11, e quando x è 11 e y è 9. Il compilatore ignora l'indentazione in un programma in C. Inoltre, il compilatore associa sempre un *else* al precedente *if* a meno che non gli venga detto di fare diversamente con l'uso di parentesi graffe {}. Dal momento che, a una prima occhiata, non potete essere sicuri di quale *if* si accoppi con un *else*, questo si definisce “problema dell'*else* sospeso”. Abbiamo eliminato l'indentazione dal seguente codice per rendere il problema più impegnativo. [Suggerimento: applicate le convenzioni sull'indentazione che avete imparato.]

- a) if ($x < 10$)
 - if ($y > 10$)
 - puts("*****");
 - else
 - puts("#####");
 - puts("\$\$\$\$\$");
- b) if ($x < 10$)
 - if ($y > 10$)
 - puts("*****");
- else
 - puts("#####");
 - puts("\$\$\$\$\$");

3.31 (*Un altro problema di else sospeso*) Modificate il seguente codice per produrre l'output mostrato. Usate le tecniche di indentazione appropriate. Non potete fare cambiamenti diversi dall'inserimento di parentesi graffe. Il compilatore ignora l'indentazione in un programma. Abbiamo eliminato l'indentazione dal seguente codice al fine di rendere il problema più impegnativo. [Nota: è possibile che non sia necessaria alcuna modifica.]

```
if (y == 8)
if (x == 5)
puts("$$$$");
else
puts("#####");
puts("$$$$");
puts("&&&&&");
```

- a) Supponendo $x = 5$ e $y = 8$, viene prodotto il seguente output.

```
#####
$$$$
&&&&
```

- b) Supponendo $x = 5$ e $y = 8$, viene prodotto il seguente output.

```
#####
$$$$
```

- c) Supponendo $x = 5$ e $y = 8$, viene prodotto il seguente output.

```
#####
&&&&
```

- d) Supponendo $x = 5$ e $y = 7$, viene prodotto il seguente output.

```
#####
$$$$
&&&&
```

3.32 (*Quadrato di asterischi*) Scrivete un programma che legga il lato di un quadrato e poi stampi quel quadrato con asterischi. Il programma deve operare con quadrati dalle dimensioni dei lati tra 1 e 20. Per esempio, se il programma legge una dimensione pari a 4, deve stampare

```
*****
***
```

3.33 (*Quadrato di asterischi vuoto*) Modificate il programma che avete scritto nel precedente esercizio in modo che stampi un quadrato vuoto. Per esempio, se il programma legge una dimensione pari a 5, deve stampare

```
*****
* * *
* * *
* * *
*****
```

3.34 (*Tester di palindromi*) Un palindromo è un numero o una frase di un testo che si legge all'indietro e in avanti. Per esempio, ognuno dei seguenti numeri interi a cinque cifre è un palindromo: 12321, 55555, 45554 e 11611. Scrivete un programma che legga un numero intero di cinque cifre e determini se sia o meno un palindromo. [Suggerimento: usate gli operatori di divisione e resto per separare il numero nelle sue cifre individuali.]

3.35 (*Stampare l'equivalente decimale di un numero binario*) Inserite un numero intero binario (di 5 cifre o meno) contenente soltanto valori pari a 0 o 1 e stampate il suo equivalente decimale. [Suggerimento: usate gli operatori di divisione e resto per ottenere le cifre del numero "binario" una alla volta da destra a sinistra. Proprio come nel sistema numerico decimale, nel quale la cifra più a destra ha un valore posizionale di 1 e la successiva cifra a sinistra ha un valore posizionale di 10, poi di 100, poi di 1.000 e così via, nel sistema numerico binario la cifra più a destra ha un valore posizionale di 1, la cifra successiva a sinistra di 2, poi di 4, poi di 8 e così via. Quindi, il numero decimale 234 si può interpretare come $4 * 1 + 3 * 10 + 2 * 100$. L'equivalente decimale del binario 1101 è $1 * 1 + 0 * 2 + 1 * 4 + 1 * 8$ ovvero $1 + 0 + 4 + 8$ ovvero 13.]

3.36 (*Quanto è veloce il vostro computer?*) Come potete davvero stabilire con quale velocità opera il vostro computer? Scrivete un programma con un ciclo while che conti da 1 a 1.000.000.000 per incrementi di uno. Ogni volta che il conto raggiunge un multiplo di 100.000.000, stampate quel numero sullo schermo. Usate il vostro orologio per cronometrare quanto tempo impiega ogni ciclo di 100 milioni di iterazioni. [Suggerimento: usate l'operatore di resto per riconoscere quando il contatore raggiunge un multiplo di 100.000.000.]

3.37 (*Trovare multipli di 10*) Scrivete un programma che stampi 100 asterischi, uno alla volta. Dopo ogni decimo asterisco, il programma deve stampare un carattere newline. [Suggerimento: contate da 1 a 100. Usate l'operatore di resto per riconoscere quando il contatore raggiunge un multiplo di 10.]

3.38 (*Contare i 7*) Scrivete un programma che legga un numero intero (di 5 cifre o meno) e determini e stampi quante cifre uguali a 7 ci sono nel numero.

3.39 (*Modello di scacchiera di asterischi*) Scrivete un programma che stampi la seguente configurazione a scacchiera:

```
*****  
* * * * * * * *  
*****  
* * * * * * * *  
*****  
* * * * * * * *  
*****  
* * * * * * * *  
*****  
* * * * * * * *
```

Il vostro programma deve usare solamente tre istruzioni di output, ciascuna rispettivamente della forma seguente:

```
printf("%s", "* ");  
printf("%s", " ");  
puts(""); // stampa un newline
```

3.40 (*Multipli di 2 con un ciclo infinito*) Scrivete un programma che continui a stampare i multipli del numero intero 2, e cioè 2, 4, 8, 16, 32, 64 e così via. Il vostro ciclo non deve terminare (ossia, dovete creare un ciclo infinito). Cosa succede quando fate eseguire questo programma?

3.41 (*Diametro, circonferenza e area di un cerchio*) Scrivete un programma che legga il raggio di un cerchio (come valore double) e calcoli e stampi il diametro, la circonferenza e l'area. Usate il valore 3,14159 per π .

3.42 Cosa c'è di sbagliato nella seguente istruzione? Riscrivetela per compiere ciò che il programmatore stava probabilmente cercando di fare.

```
printf("%d", ++(x + y));
```

3.43 (Lati di un triangolo) Scrivete un programma che legga tre valori interi diversi da zero e determini e stampi se essi possono rappresentare i lati di un triangolo.

3.44 (Lati di un triangolo rettangolo) Scrivete un programma che legga tre numeri interi diversi da zero e determini e stampi se possono essere i lati di un triangolo rettangolo.

3.45 (Fattoriale) Il fattoriale di un numero intero non negativo n si scrive $n!$ (pronunciato “ n fattoriale”) ed è definito come segue:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 1 \quad (\text{per valori di } n \text{ maggiori o uguali a 1})$$

e

$$n! = 1 \quad (\text{per } n = 0).$$

Per esempio, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, che è 120.

- a) Scrivete un programma che legga un numero intero non negativo e calcoli e stampi il suo fattoriale.
- b) Scrivete un programma che valuti il valore della costante matematica e usando la formula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

- c) Scrivete un programma che calcoli il valore di e^x usando la formula:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

3.46 (Crescita della popolazione mondiale) La popolazione mondiale è cresciuta considerevolmente nel corso dei secoli. La crescita continua potrebbe col tempo portare al raggiungimento dei limiti dell'aria respirabile, dell'acqua potabile, dei terreni coltivabili e di altre risorse. È dimostrato che da qualche anno c'è un rallentamento della crescita e che la popolazione mondiale potrebbe raggiungere un picco nel corso del secolo, per poi iniziare a calare.

Per questo esercizio, fate una ricerca sui temi relativi alla crescita della popolazione mondiale. Si tratta di un argomento controverso, quindi assicuratevi di prendere in esame vari punti di vista. Ottenete stime sull'attuale popolazione mondiale e sul suo tasso di crescita. Scrivete un programma che calcoli la crescita della popolazione mondiale ogni anno per i prossimi 100 anni, *usando il presupposto semplificativo che l'attuale tasso di crescita resterà costante*. Stampate i risultati in una tabella. La prima colonna dovrebbe visualizzare l'anno (da 1 a 100), la seconda la popolazione mondiale prevista per la fine di quell'anno, e la terza l'aumento numerico nella popolazione mondiale che si verificherebbe quell'anno. Usando i vostri risultati, determinate gli anni in cui la popolazione diventerebbe il doppio e poi il quadruplo di oggi.

3.47 (Forzare la privacy con la crittografia) La crescita esplosiva delle comunicazioni via Internet e l'archiviazione dei dati su computer connessi a Internet ha notevolmente aumentato i problemi di privacy. Il campo della crittografia riguarda la codifica dei dati per renderne difficile (e si spera, con gli schemi più avanzati, impossibile) la lettura agli utenti non autorizzati. In questo esercizio esaminerete uno schema semplice per crittografare e decrittografare i dati. Una società che desidera inviare dati su Internet vi ha chiesto di scrivere un programma per crittografarli e poterli trasmettere in modo più sicuro. Tutti i dati vengono trasmessi come numeri interi a quattro cifre. L'applicazione deve leggere un numero intero di quattro cifre inserito dall'utente e crittografarlo come segue: sostituire ogni cifra con il resto della divisione per 10 dello stesso numero al quale è stato prima sommato il valore 7; scambiare la prima cifra con la terza e poi la seconda con la quarta; stampare quindi il numero intero crittografato. Scrivete un'applicazione separata che legge un numero intero a quattro cifre crittografato e lo decodifica (invertendo lo schema di crittografia) per formare il numero originale. [Progetto di lettura opzionale: nelle applicazioni a livello industriale, utilizzerete tecniche di crittografia molto più potenti di quelle presentate in questo esercizio. Cercate “crittografia a chiave pubblica” in generale e lo schema a chiave pubblica specifico PGP (*Pretty Good Privacy*). Potreste anche voler studiare lo schema RSA, che è ampiamente usato in applicazioni a livello industriale.]

CAPITOLO

4

Sommario del capitolo

- 4.1 Introduzione
- 4.2 Aspetti essenziali dell'iterazione
- 4.3 Iterazione controllata da contatore
- 4.4 Istruzione di iterazione `for`
- 4.5 Esempi di uso dell'istruzione `for`
- 4.6 Istruzione di selezione multipla `switch`
- 4.7 Istruzione di iterazione `do...while`
- 4.8 Istruzioni `break` e `continue`
- 4.9 Operatori logici
- 4.10 Confondere gli operatori di uguaglianza (`==`) e di assegnazione (`=`)
- 4.11 Sintesi della programmazione strutturata
- 4.12 Programmazione sicura in C
- 4.13 Riepilogo

Controllo nei programmi

Obiettivi

- Imparare gli aspetti essenziali dell'iterazione controllata da contatore.
- Usare le istruzioni di iterazione `for` e `do...while` per eseguire ripetutamente delle istruzioni.
- Comprendere la selezione multipla con l'istruzione di selezione `switch`.
- Usare le istruzioni `break` e `continue` per alterare il flusso di controllo.
- Usare gli operatori logici per formare condizioni complesse nelle istruzioni di controllo.
- Evitare le conseguenze derivanti dal confondere operatori di uguaglianza e operatori di assegnazione.

4.1 Introduzione

Ormai dovreste essere a vostro agio con la lettura e la scrittura di semplici programmi in C. Ora considereremo con maggiore dettaglio l'iterazione e presenteremo le istruzioni di iterazione `for` e `do...while` del C. Introdurremo anche:

- l'istruzione di selezione multipla `switch`;
- l'istruzione `break` per uscire immediatamente da certe istruzioni di controllo;
- l'istruzione `continue` per saltare il resto del corpo di un'istruzione di iterazione e procedere con la successiva iterazione del ciclo.

Esamineremo anche gli operatori logici usati per combinare delle condizioni e riassumeremo i principi della programmazione strutturata presentati nel corso di questo capitolo e del precedente.

4.2 Aspetti essenziali dell'iterazione

La maggior parte dei programmi si basa sull'iterazione, ovvero su cicli. Un ciclo (*loop*) è un gruppo di istruzioni che il computer esegue ripetutamente finché una qualche **condizione di continuazione del ciclo** rimane vera. Abbiamo esaminato due modalità di iterazione:

1. Iterazione controllata da contatore.
2. Iterazione controllata da sentinella.

Avete visto nel Capitolo 3 che l'iterazione controllata da contatore usa una **variabile di controllo** per contare il numero di iterazioni per un gruppo di istruzioni da eseguire. Quando il valore della variabile di controllo indica che è stato eseguito il numero corretto di iterazioni, il ciclo termina e l'esecuzione continua con l'istruzione dopo l'istruzione di iterazione.

Sempre nel Capitolo 3, avete visto che usiamo i valori sentinella per controllare l'iterazione se non si conosce in anticipo il numero preciso di iterazioni, e il ciclo comprende istruzioni che leggono dati ogni volta che viene eseguito il ciclo. Un valore sentinella indica la "fine dei dati". La sentinella viene inserita dopo che tutti i dati regolari sono stati forniti al programma. Le sentinelle devono essere distinte dai dati regolari.

✓ Autovalutazione

1. (*Completare*) Un ciclo è un gruppo di istruzioni che il computer esegue ripetutamente finché una condizione di _____ rimane *vera*.

Risposta: continuazione del ciclo.

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) I valori sentinella sono usati per controllare l'iterazione quando non si conosce in anticipo il numero preciso di iterazioni, e il ciclo comprende istruzioni che leggono dati ogni volta che viene eseguito il ciclo.
- b) Il valore sentinella indica la "fine dei dati".
- c) La sentinella è inserita dopo che sono stati forniti tutti i dati regolari.
- d) La sentinella deve corrispondere a dati regolari.

Risposta: d) è *falsa*. In realtà, la sentinella deve essere distinta dai dati regolari.

4.3 Iterazione controllata da contatore

L'iterazione controllata da contatore richiede:

- il **nome** di una variabile di controllo;
- il **valore iniziale** della variabile di controllo;
- l'**incremento** (o il **decremento**) con cui la variabile di controllo è modificata in ogni iterazione;
- la condizione di continuazione del ciclo che verifica il **valore finale** della variabile di controllo per determinare se il ciclo deve continuare.

Considerate la Figura 4.1, che stampa i numeri da 1 a 5. La definizione

```
int counter = 1; // inizializzazione
```

dà il nome alla variabile di controllo (counter), la definisce come un numero intero, le riserva spazio nella memoria e imposta il suo valore iniziale a 1.

```

1 // fig04_01.c
2 // Iterazione controllata da contatore.
3 #include <stdio.h>
4
5 int main(void) {
6     int counter = 1; // inizializzazione
7
8     while (counter <= 5) { // condizione di iterazione
9         printf("%d ", counter);
10        ++counter; // incremento
11    }
12}
```

```
13     puts("");
14 }
```

```
1 2 3 4 5
```

Figura 4.1 Iterazione controllata da contatore.

L'istruzione

```
++counter; // incremento
```

incrementa `counter` di 1 alla fine di ogni iterazione del ciclo. La condizione di `while`

```
counter <= 5
```

verifica se il valore della variabile di controllo è minore o uguale a 5 (l'ultimo valore per il quale la condizione è vera). Il ciclo nell'istruzione `while` termina quando la variabile di controllo supera il valore 5 (cioè il contatore diventa 6).

Usare contatori con numeri interi

I valori in virgola mobile possono essere approssimati, pertanto il controllo con contatore dei cicli con variabili in virgola mobile può portare a valori del contatore e a test di terminazione imprecisi. Per questo motivo, dovreste sempre controllare i cicli usando valori numerici interi.

✓ Autovalutazione

- (Scelta multipla) Quale tra le seguenti indicazioni viene richiesta dall'iterazione controllata da contatore?
 - Il nome e il valore iniziale di una variabile di controllo (o contatore del ciclo).
 - L'incremento (o il decremento) con cui la variabile di controllo è modificata ogni volta nel corso del ciclo.
 - La condizione di continuazione del ciclo che verifica il valore finale della variabile di controllo per determinare se il ciclo deve continuare.
 - L'iterazione controllata da contatore richiede tutto quanto sopra elencato.

Risposta: d.

- (Scelta multipla) In riferimento al programma in questo paragrafo, quale delle seguenti affermazioni a), b) o c) è falsa?

- La variabile di controllo `counter` viene incrementata di 1 a ogni iterazione del ciclo.
- Il ciclo termina quando il valore di `counter` è 5.
- Il corpo dell'istruzione `while` viene eseguito anche quando il valore della variabile di controllo è 5.
- Tutte le affermazioni precedenti sono vere.

Risposta: b) è falsa. In realtà, il ciclo termina quando il valore della variabile di controllo diventa 6.

4.4 Istruzione di iterazione for

L'istruzione di iterazione `for` (righe 8-10 della Figura 4.2) gestisce tutti i dettagli dell'iterazione controllata da contatore. Per una migliore leggibilità, cercate di inserire l'intestazione dell'istruzione `for` (riga 8) in un'unica riga. L'esecuzione dell'istruzione `for` avviene come segue.

- Quando l'istruzione `for` inizia l'esecuzione, definisce la variabile di controllo `counter` e la inizializza a 1.
- Verifica poi la condizione di continuazione del ciclo `counter <= 5`. Il valore iniziale di `counter` è 1, quindi la condizione è *vera*, e l'istruzione `for` esegue la sua istruzione `printf` (riga 9) e stampa il valore di `counter`, ossia 1.

- In seguito, l'istruzione `for` incrementa la variabile di controllo `counter` con l'espressione `++counter`, quindi verifica nuovamente la condizione di continuazione del ciclo. La variabile di controllo è ora uguale a 2, quindi la condizione è ancora *vera*, e l'istruzione `for` esegue un'altra volta la sua istruzione `printf`.
- Questo processo continua finché la variabile di controllo `counter` diventa 6. A questo punto, la condizione di continuazione del ciclo è *falsa* e l'iterazione termina.

Il programma continua eseguendo la prima istruzione dopo l'istruzione `for` (riga 12).

```

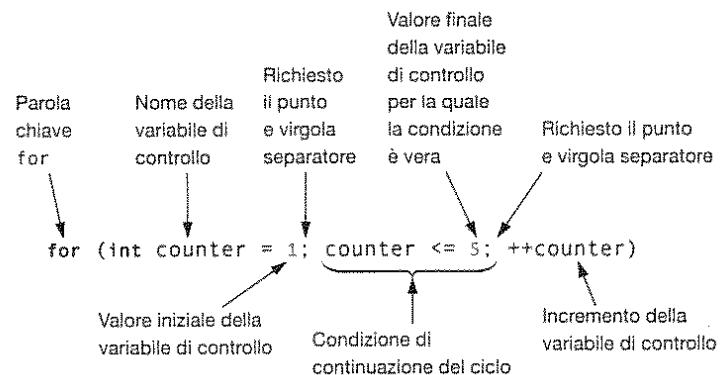
1 // fig04_02.c
2 // Iterazione controllata da contatore con l'istruzione for.
3 #include <stdio.h>
4
5 int main(void) {
6     // inizializzazione, condizione dell'iterazione e incremento
7     // sono tutti inclusi nell'intestazione dell'istruzione for.
8     for (int counter = 1; counter <= 5; ++counter) {
9         printf("%d ", counter);
10    }
11
12    puts(""); // stampa un newline
13 }
```

1 2 3 4 5

Figura 4.2 Iterazione controllata da contatore con l'istruzione `for`.

Componenti dell'intestazione dell'istruzione `for`

Il diagramma seguente illustra in maggiore dettaglio l'istruzione `for` della Figura 4.2, che specifica ognuno degli elementi necessari per l'iterazione controllata da contatore. Se vi è più di un'istruzione nel corpo del `for`, sono necessarie le parentesi graffe. Come con le altre istruzioni di controllo, dovreste sempre inserire il corpo di un'istruzione `for` tra parentesi graffe, anche se contiene un'unica istruzione.



Le variabili di controllo definite in un'intestazione del `for` esistono solo fino al termine del ciclo

Quando definite la variabile di controllo nell'intestazione del `for` prima del primo punto e virgola (;), come nella riga 8 della Figura 4.2:

```
for (int counter = 1; counter <= 5; ++counter) {
```

 la variabile di controllo esiste solo fino al termine del ciclo. Quindi, il tentativo di accesso alla variabile di controllo dopo la parentesi graffa destra di chiusura () dell'istruzione for è un errore di compilazione.

Errori di tipo off-by-one

 Se avessimo scritto la condizione di continuazione del ciclo counter ≤ 5 come counter < 5 , il ciclo verrebbe eseguito solo quattro volte. Questo è un comune errore logico chiamato **errore di tipo off-by-one** (letteralmente “sfasamento di uno”). L'uso del valore finale di una variabile di controllo nella condizione di un'istruzione while o di un'istruzione for assieme all'operatore relazionale \leq può contribuire a evitare gli errori di tipo off-by-one. Per stampare i valori da 1 a 5, per esempio, la condizione di continuazione del ciclo deve essere counter ≤ 5 invece che counter < 6 .

Formato generale di un'istruzione for

Il formato generale dell'istruzione for è

```
for (inizializzazione; condizioneDiContinuazioneDelCiclo; incremento) {
    istruzione
}
```

dove

- *inizializzazione* dà il nome alla variabile di controllo del ciclo e imposta il suo valore iniziale;
- *condizioneDiContinuazioneDelCiclo* determina se l'esecuzione del ciclo deve continuare;
- *incremento* modifica il valore della variabile di controllo dopo l'esecuzione dell'*istruzione* così che alla fine la condizione di continuazione del ciclo diventa falsa.

I due punti e virgola nell'intestazione dell'istruzione for sono necessari. Se la condizione di continuazione del ciclo inizialmente è falsa, il programma non esegue il corpo dell'istruzione for, e procede invece con l'esecuzione dell'istruzione successiva al for.

 I cicli infiniti si verificano quando la condizione di continuazione del ciclo non diventa mai falsa. Per evitare cicli infiniti bisogna accertarsi che non ci sia un punto e virgola immediatamente dopo l'intestazione di un'istruzione while. In un ciclo controllato da contatore bisogna accertarsi che la variabile di controllo sia incrementata (o decrementata) così che alla fine la condizione di continuazione del ciclo diventi falsa. In un ciclo controllato da sentinella bisogna accertarsi che il valore sentinella sia infine inserito.

Le espressioni nell'intestazione dell'istruzione for sono opzionali

Le tre espressioni nell'intestazione di un'istruzione for sono opzionali. Se viene omessa la *condizioneDiContinuazioneDelCiclo*, la condizione viene considerata sempre vera, generando così un ciclo infinito. Potete omettere l'espressione *inizializzazione* se il programma inizializza la variabile di controllo prima del ciclo. Potete omettere l'espressione *incremento* se il programma calcola l'incremento nel corpo del ciclo o se non è necessario alcun incremento.

L'espressione di incremento agisce come un'istruzione separata

L'espressione *incremento* nell'istruzione for agisce come un'istruzione separata in C alla fine del corpo del for. Quindi, le seguenti espressioni sono tutte equivalenti quando sono poste nell'espressione di incremento di un'istruzione for:

```
counter = counter + 1
counter += 1
++counter
counter++
```

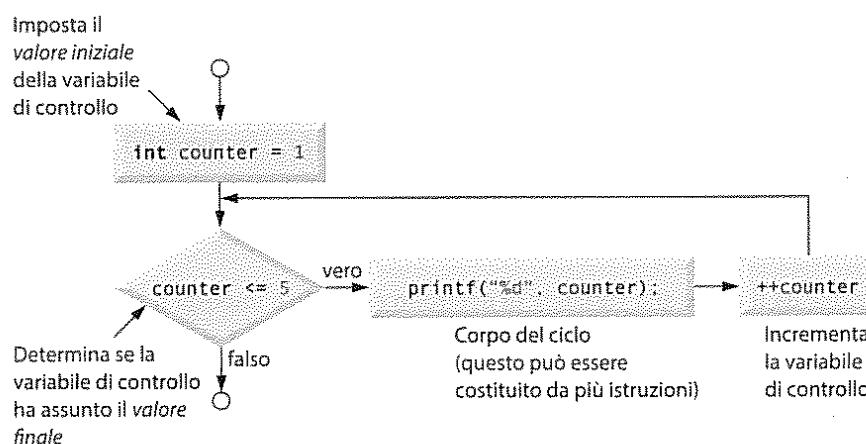
L'incremento nell'espressione *incremento* di un'istruzione for può essere negativo (nel qual caso è in realtà un *decremento* e il ciclo conta *all'indietro*).

Usare la variabile di controllo di un'istruzione for nel corpo dell'istruzione

Spesso i programmi stampano il valore della variabile di controllo o lo usano nei calcoli all'interno del corpo di un ciclo, ma questo non è necessario. È comune usare la variabile di controllo per controllare l'iterazione, senza menzionarla mai nel corpo dell'istruzione for. Sebbene sia possibile modificare il valore della variabile di controllo nel corpo di un ciclo for, evitate questa pratica perché può portare a errori subdoli. È meglio non modificarlo.

**Diagramma di flusso dell'istruzione for**

Sotto è illustrato il diagramma di flusso per l'istruzione for della Figura 4.2:



Questo diagramma di flusso rende evidente che l'inizializzazione avviene solo una volta e che l'incremento è effettuato *dopo* ogni esecuzione dell'istruzione del corpo.

✓ Autovalutazione

- (Vero/Falso)** Quando definite la variabile di controllo nell'intestazione del for prima del primo punto e virgola (:), la variabile di controllo esiste solo fino al termine del ciclo.

Risposta: Vero.

- (Scelta multipla)** Quali delle seguenti affermazioni a), b) o c) è *vera*?

- L'intestazione dell'istruzione for specifica ognuno degli elementi necessari per l'iterazione controllata da contatore con una variabile di controllo.
- Se vi è più di un'istruzione nel corpo del for, sono necessarie le parentesi graffe.
- Dovreste sempre inserire il corpo di un'istruzione di controllo tra parentesi graffe, anche se contiene un'unica istruzione.
- Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

4.5 Esempi di uso dell'istruzione for

I seguenti esempi mostrano alcuni metodi con cui far variare la variabile di controllo in un'istruzione for.

- Far variare la variabile di controllo da 1 a 100 per incrementi di 1.

```
for (int i = 1; i <= 100; ++i)
```

- Far variare la variabile di controllo da 100 a 1 per incrementi di -1 (cioè, *decrementi* di 1).

```
for (int i = 100; i >= 1; --i)
```

3. Far variare la variabile di controllo da 7 a 77 per incrementi di 7.

```
for (int i = 7; i <= 77; i += 7)
```

4. Far variare la variabile di controllo da 20 a 2 per incrementi di -2.

```
for (int i = 20; i >= 2; i -= 2)
```

5. Far variare la variabile di controllo secondo la seguente sequenza di valori: 2, 5, 8, 11, 14 e 17.

```
for (int j = 2; j <= 17; j += 3)
```

6. Far variare la variabile di controllo secondo la seguente sequenza di valori: 44, 33, 22, 11, 0.

```
for (int j = 44; j >= 0; j -= 11)
```

Applicazione: sommare i numeri interi pari da 2 a 100

Il programma della Figura 4.3 usa l'istruzione for per sommare tutti i numeri interi pari da 2 a 100. Ogni iterazione del ciclo (righe 8-10) aggiunge il valore corrente della variabile di controllo number alla variabile sum.

```
1 // fig04_03.c
2 // Somma con for.
3 #include <stdio.h>
4
5 int main(void) {
6     int sum = 0; // inizializza sum
7
8     for (int number = 2; number <= 100; number += 2) {
9         sum += number; // aggiungi number a sum
10    }
11
12    printf("Sum is %d\n", sum);
13 }
```

Sum is 2550

Figura 4.3 Somma con for.

Applicazione: calcolo dell'interesse composto

Il prossimo esempio calcola l'interesse composto con l'uso dell'istruzione for. Considerate la seguente enunciazione del problema:

Una persona investe \$1000,00 in un conto corrente che frutta il 5% di interesse. Supponendo che l'intero interesse resti depositato nel conto, calcolate e stampate la quantità di denaro nel conto alla fine di ogni anno per 10 anni. Usate la seguente formula per determinare queste quantità:

$$a = p(1 + r)^n$$

dove

p è la quantità iniziale di denaro investita (cioè, il capitale, che qui è \$1000.00),

r è il tasso annuale di interesse (per esempio, 0.05 per 5%),

n è il numero degli anni, che qui è 10, e

a è la quantità di denaro in deposito alla fine dell'anno n.

Nella soluzione di questo problema (Figura 4.4) viene usato un ciclo controllato da contatore per eseguire il calcolo indicato per ciascuno dei 10 anni in cui il denaro rimane in deposito. L'istruzione for esegue il corpo del ciclo 10 volte, facendo variare la variabile di controllo da 1 a 10 per incrementi di 1. Il C *non* include un operatore esponenziale, ma possiamo usare a questo scopo la funzione pow (riga 17) della Libreria Standard. La chiamata pow(x, y) calcola il valore di x elevato alla potenza y. La funzione riceve due argomenti di tipo

double. Quando ha completato il calcolo, pow restituisce un valore double, che viene poi moltiplicato per principal (riga 17).

```

1 // fig04_04.c
2 // Calcolo dell'interesse composto.
3 #include <stdio.h>
4 #include <math.h>
5
6 int main(void) {
7     double principal = 1000.0; // capitale iniziale
8     double rate = 0.05; // tasso di interesse annuale
9
10    // stampa le intestazioni delle colonne della tabella
11    printf("%4s%21s\n", "Year", "Amount on deposit");
12
13    // calcola la quantità in deposito per ognuno dei dieci anni
14    for (int year = 1; year <= 10; ++year) {
15
16        // calcola la nuova quantità per l'anno specificato
17        double amount = principal * pow(1.0 + rate, year);
18
19        // stampa una riga della tabella
20        printf("%4d%21.2f\n", year, amount);
21    }
22 }
```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

Figura 4.4 Calcolo dell'interesse composto.

È necessario includere l'intestazione `<math.h>` (riga 4) per usare la funzione `pow` e le altre funzioni matematiche del C.¹ Se l'intestazione non viene inclusa, questo programma funzionerebbe male, poiché il linker non sarebbe in grado di trovare la funzione `pow`. La funzione `pow` richiede due argomenti `double`, ma la variabile `year` è un intero. Il file `math.h` include informazioni che dicono al compilatore di convertire il valore di `year` in una rappresentazione temporanea `double` prima di chiamare la funzione. Queste informazioni sono contenute nel **prototipo di funzione** di `pow`. I prototipi di funzioni sono spiegati nel Capitolo 5, dove sono presentate molte altre funzioni della libreria `math`.

1. Con il compilatore `gcc`, dovete includere l'opzione `-lm` (es. `gcc -lm fig04_04.c`) quando compilate il programma della Figura 4.4. Questa opzione collega la libreria `math` al programma.

Formattare output numerici

Nel programma è usata la specifica di conversione `%21.2f` per stampare il valore della variabile `amount`. Il `21` nella specifica di conversione indica la **larghezza di campo** in cui il valore sarà stampato. Una larghezza di campo di 21 specifica che il valore stampato apparirà in 21 posizioni di carattere. Come avete appreso nel Capitolo 3, `.2` specifica la precisione (cioè il numero di posizioni decimali). Se il numero di caratteri stampati è minore della larghezza di campo, il valore sarà allineato a destra con spazi iniziali. Questo è particolarmente utile per allineare i punti decimali di valori in virgola mobile verticalmente. Per allineare a sinistra un valore in un campo, mettete un `-` (segno meno) tra il simbolo `%` e la larghezza di campo. Esamineremo in dettaglio le potenti capacità di formattazione di `printf` e `scanf` nel Capitolo 9.

Precisione e requisiti di memoria dei numeri in virgola mobile

Di norma, le variabili di tipo `float` richiedono quattro byte di memoria per circa 7 cifre significative, mentre le variabili di tipo `double` richiedono otto byte di memoria per circa 15 cifre significative (hanno una precisione pressoché doppia rispetto alle variabili `float`). La maggior parte dei programmati preferisce utilizzare il tipo `double`. Per default, il C tratta i valori in virgola mobile (es. 3,14159) come tipo `double`. Questi valori nel codice sorgente vengono chiamati **letterali in virgola mobile**.

In C troviamo anche variabili di tipo `long double`, che in genere richiedono 12 o 16 byte di memoria. Il C standard stabilisce le dimensioni minime di ciascun tipo in virgola mobile e specifica che il tipo `double` fornisce almeno la stessa precisione di `float` e che il tipo `long double` fornisce almeno la stessa precisione di `double`. Per un elenco dei tipi numerici fondamentali del C e delle loro caratteristiche, consultate

https://en.cppreference.com/w/c/language/arithmetic_types

I numeri in virgola mobile sono approssimazioni

Nell'aritmetica convenzionale, i numeri in virgola mobile spesso si presentano come risultato di una divisione: quando dividiamo 10 per 3, il risultato è 3,333333... con la sequenza dei 3 che si ripete all'infinito. Il computer assegna solo una quantità fissata di spazio per contenere un tale valore, così il valore in virgola mobile memorizzato può essere soltanto un'approssimazione. Possiamo quindi dire che i numeri in virgola mobile risentono di quello che viene definito un **errore di rappresentazione**. Presumere che essi siano rappresentati in maniera precisa (per es. usandoli nei confronti di uguaglianza) può portare a risultati scorretti.

I numeri in virgola mobile hanno numerose applicazioni, soprattutto per valori di misura. Per esempio, quando parliamo di una "normale" temperatura corporea di 98,6 gradi Fahrenheit, non abbiamo bisogno di essere precisi fino a un grande numero di cifre. Quando leggiamo la temperatura su un termometro come 98,6, essa può in realtà essere 98,5999473210643. Indicare questo numero semplicemente come 98,6 va benissimo per la maggior parte delle applicazioni relative alla temperatura corporea.

Un avvertimento relativo alla stampa di valori arrotondati

In questo esempio abbiamo definito di tipo `double` le variabili `amount`, `principal` e `rate`. Stiamo trattando con parti frazionarie di dollari e quindi ci serve un tipo che permetta di avere valori con punti decimali. Purtroppo però i numeri in virgola mobile possono causare problemi. Ecco una spiegazione semplice di cosa può andare storto quando si usano i numeri in virgola mobile per rappresentare quantità monetarie con due cifre alla destra del punto decimale. Due importi in dollari memorizzati nel sistema potrebbero essere 14.234 (arrotondato a 14.23 per la stampa) e 18.673 (arrotondato a 18.67 per la stampa). Quando questi importi vengono sommati, producono come risultato 32.907, arrotondato a 32.91 per la stampa. Quindi la vostra stampa potrebbe apparire in questo modo

$$\begin{array}{r} 14.23 \\ + 18.67 \\ \hline 32.91 \end{array}$$

ma se si sommano i singoli numeri stampati, il risultato è 32.90. Siete stati avvisati!

Numeri in virgola mobile: errori di rappresentazione anche con semplici importi in dollari
Ci possono essere errori di rappresentazione anche con semplici importi (come quelli su uno scontrino del supermercato o sul conto di un ristorante) nel caso questi vengano memorizzati come tipi double. Per evidenziare questo aspetto, abbiamo creato un semplice programma con la dichiarazione

```
double d = 123.02;
```

e poi abbiamo stampato il valore di d con molte cifre di precisione alla destra del punto decimale. L'output mostra il valore 123.02 come 123.019999..., un altro esempio di errore di rappresentazione. Sebbene alcuni importi possano essere rappresentati con precisione come double, in molti altri casi ciò non avviene. Si tratta di un problema comune in diversi linguaggi di programmazione.

✓ Autovalutazione

1. (*Completare*) Per _____ un valore in un campo, si mette un - (segno meno) tra il simbolo % e la larghezza di campo.

Risposta: allineare a sinistra.

2. (*Scelta multipla*) Quale tra le seguenti intestazioni dell'istruzione for non è corretta?

- a) Far variare la variabile di controllo da 1 a 100 per incrementi di 1.

```
for (int i = 1; i <= 100; ++i)
```

- b) Far variare la variabile di controllo da 7 a 77 per incrementi di 7.

```
for (int i = 7; i <= 77; i += 7)
```

- c) Far variare la variabile di controllo secondo la seguente sequenza di valori: 2, 5, 8, 11, 15, 17.

```
for (int j = 2; j <= 17; j += 3)
```

- d) Far variare la variabile di controllo da 20 a 2 per incrementi di -2.

```
for (int i = 20; i >= 2; i -= 2)
```

Risposta: c) non è corretta. L'istruzione for in realtà genera la sequenza 2, 5, 8, 11, 14, 17, senza il valore 15.

4.6 Istruzione di selezione multipla switch

Nel Capitolo 3 abbiamo esaminato l'istruzione di selezione singola if e l'istruzione di selezione doppia if... else. A volte, un algoritmo conterrà una serie di decisioni in cui una variabile o un'espressione viene testata separatamente per ognuno dei valori interi che può assumere e per i quali vengono intraprese azioni differenti. Tale situazione è chiamata selezione multipla. Il C fornisce l'istruzione di selezione multipla switch per trattare questo processo decisionale.

L'istruzione switch consiste in una serie di etichette case, un caso default opzionale e istruzioni da eseguire per ognuno dei case. La Figura 4.5 usa switch per contare il numero di volte che ogni singolo voto a lettera² è stato ottenuto dagli studenti in un esame.

```
1 // fig04_05.c
2 // Conteggio di voti a lettera con switch.
3 #include <stdio.h>
4
5 int main(void) {
6     int aCount = 0;
7     int bCount = 0;
```

2. I voti a lettera, tipici delle scuole statunitensi, sono espressi con le lettere dell'alfabeto "a", "b", "c", "d" e "f". (N.d.T.)

```
8     int cCount = 0;
9     int dCount = 0;
10    int fCount = 0;
11
12    puts("Enter the letter grades.");
13    puts("Enter the EOF character to end input.");
14    int grade = 0; // un voto
15
16    // ripeti finche' l'utente non immette la sequenza di end-of-file
17    while ((grade = getchar()) != EOF) {
18
19        // determina quale voto e' stato inserito
20        switch (grade) { // switch annidato nel while
21            case 'A': // il voto e' la lettera maiuscola A
22            case 'a': // o la lettera minuscola a
23                ++aCount;
24                break; // necessario per uscire dallo switch
25            case 'B': // il voto e' la lettera maiuscola B
26            case 'b': // o la lettera minuscola b
27                ++bCount;
28                break;
29            case 'C': // il voto e' la lettera maiuscola C
30            case 'c': // o la lettera minuscola c
31                ++cCount;
32                break;
33            case 'D': // il voto e' la lettera maiuscola D
34            case 'd': // o la lettera minuscola d
35                ++dCount;
36                break;
37            case 'F': // il voto e' la lettera maiuscola F
38            case 'f': // o la lettera minuscola f
39                ++fCount;
40                break;
41            case '\n': // ignora i newline,
42            case '\t': // le tabulazioni
43            case ' ': // e gli spazi in input
44                break;
45            default // cattura tutti gli altri caratteri
46                printf("%s", "Incorrect letter grade entered.");
47                puts(" Enter a new grade.");
48                break; // opzionale; uscira' comunque dallo switch
49        } // fine di switch
50    } // fine di while
51
52    // stampa il riepilogo dei risultati
53    puts("\nTotals for each letter grade are:");
54    printf("A: %d\n", aCount);
55    printf("B: %d\n", bCount);
56    printf("C: %d\n", cCount);
57    printf("D: %d\n", dCount);
58    printf("F: %d\n", fCount);
59 }
```

```

Enter the letter grades.
Enter the EOF character to end input.
a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z — Non tutti i sistemi mostrano una rappresentazione del carattere EOF.

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1

```

Figura 4.5 Conteggio di voti a lettera con switch.

Input di caratteri

Nel programma, l'utente inserisce i voti a lettera degli studenti. Nell'intestazione del while (riga 17),

```
while ((grade = getchar()) != EOF)
```

viene eseguita per prima l'assegnazione fra parentesi (`grade = getchar()`). La funzione `getchar` (da `<stdio.h>`) legge un carattere dalla tastiera e lo memorizza nella variabile intera `grade`. I caratteri sono normalmente memorizzati in variabili di tipo `char`. Tuttavia, il C può memorizzare caratteri in variabili di qualsiasi tipo intero perché di solito sono rappresentati nel computer come interi di un solo byte. La funzione `getchar` restituisce come `int` il carattere immesso dall'utente. Possiamo trattare un carattere o come un intero o come un carattere, a seconda del suo uso. Per esempio, l'istruzione

```
printf("The character (%c) has the value %d.\n", 'a', 'a');
```

usa le specifiche di conversione `%c` e `%d` per stampare '`a`' e il suo valore intero. Il risultato è

```
The character (a) has the value 97.
```

Si possono leggere i caratteri con `scanf` usando la specifica di conversione `%c`. Il numero intero 97 è la rappresentazione numerica del carattere '`a`' nel computer. Molti computer oggi usano l'insieme di caratteri Unicode®. Nell'Appendice B trovate l'**insieme caratteri ASCII (American Standard Code for Information Interchange)** con i suoi valori numerici. ASCII è un sottoinsieme di Unicode.

Le assegnazioni hanno valori

L'intera operazione di assegnazione ha in realtà un valore. Il valore dell'espressione di assegnazione `grade = getchar()` è il carattere restituito da `getchar` e assegnato alla variabile `grade`. Il fatto che le assegnazioni abbiano valori può essere utile per assegnare a diverse variabili lo stesso valore. Per esempio,

```
a = b = c = 0;
```

esegue dapprima l'assegnazione `c = 0` (poiché l'operatore `=` è associativo da destra a sinistra). Alla variabile `b` è allora assegnato il valore dell'assegnazione `c = 0` (che è 0). Quindi, alla variabile `a` è assegnato il valore dell'assegnazione `b = (c = 0)` (che è pure 0).

Il valore dell'assegnazione `grade = getchar()` viene confrontato con il valore di `EOF` (un simbolo che indica la fine di un file e il cui acronimo sta per "end of file"). Usiamo `EOF` (che normalmente ha il valore `-1`) come valore sentinella. L'utente digita una combinazione di tasti dipendente dal sistema per indicare "end of file", ossia "non ho più dati da inserire". `EOF` è una costante intera simbolica definita nell'intestazione `<stdio.h>` (vedremo nel Capitolo 6 come sono definite le costanti simboliche). Se il valore assegnato a `grade` è uguale a `EOF`, il programma termina.

Abbiamo scelto di rappresentare in questo programma i caratteri con il tipo `int` perché `EOF` ha un valore intero (come si è detto, normalmente `-1`). Effettuare il test sulla costante simbolica `EOF`, piuttosto che `-1`, rende i programmi più portabili. Il C standard stabilisce che `EOF` è un valore intero negativo (ma non necessariamente `-1`). Quindi, i valori di `EOF` possono variare a seconda del sistema in uso.

Inserimento dell'indicatore EOF

La combinazione di tasti per inserire `EOF` (end of file) è dipendente dal sistema. Sui sistemi Linux/UNIX/macOS, l'indicatore `EOF` si inserisce digitando su una riga separata

Ctrl + d

Questa notazione indica di premere simultaneamente il tasto `Ctrl` e il tasto `d`. Su altri sistemi, come Windows di Microsoft, l'indicatore `EOF` può essere inserito digitando

Ctrl + z

Su Windows dovete inoltre premere *Invio*.

L'utente inserisce i voti attraverso la tastiera; quando preme il tasto *Invio*, i caratteri sono letti dalla funzione `getchar` uno alla volta. Se il carattere inserito non è uguale a `EOF`, viene eseguita l'istruzione `switch` (righe 20-49).

Dettagli dell'istruzione switch

La parola chiave `switch` è seguita dal nome della variabile `grade` tra parentesi. Questa è chiamata **espressione di controllo**. Il valore di questa espressione è confrontato con ognuna delle **etichette case**. Ogni `case` può avere una o più azioni, ma non sono necessarie le parentesi graffe attorno alle sequenze di azioni in ogni `case`.

Supponete che l'utente abbia inserito la lettera `C` come valore per `grade`. Quando lo `switch` confronta `C` con ogni `case`, se si verifica una corrispondenza (`case 'C':`), vengono eseguite le istruzioni per quel `case`. Nel caso della lettera `C`, lo `switch` incrementa `cCount` di 1 (riga 31), quindi l'istruzione `break` (riga 32) causa l'uscita immediata dallo `switch` e fa in modo che il controllo del programma continui con la prima istruzione dopo l'istruzione `switch`.

Utilizziamo in questo caso l'istruzione `break` perché i vari `case` in un'istruzione `switch` verrebbero altrimenti eseguiti tutti insieme. In mancanza di istruzioni `break`, ogni volta che si verificherà un confronto positivo verranno eseguite le istruzioni per tutti i `case` rimanenti. (Questa caratteristica è usata raramente, sebbene sia perfetta per scrivere in modo compatto il programma dell'Esercizio 4.38, la canzone ripetitiva *The Twelve Days of Christmas!*) Dimenticare un'istruzione `break` quando è necessaria in un'istruzione `switch` è un errore logico.



Il caso default

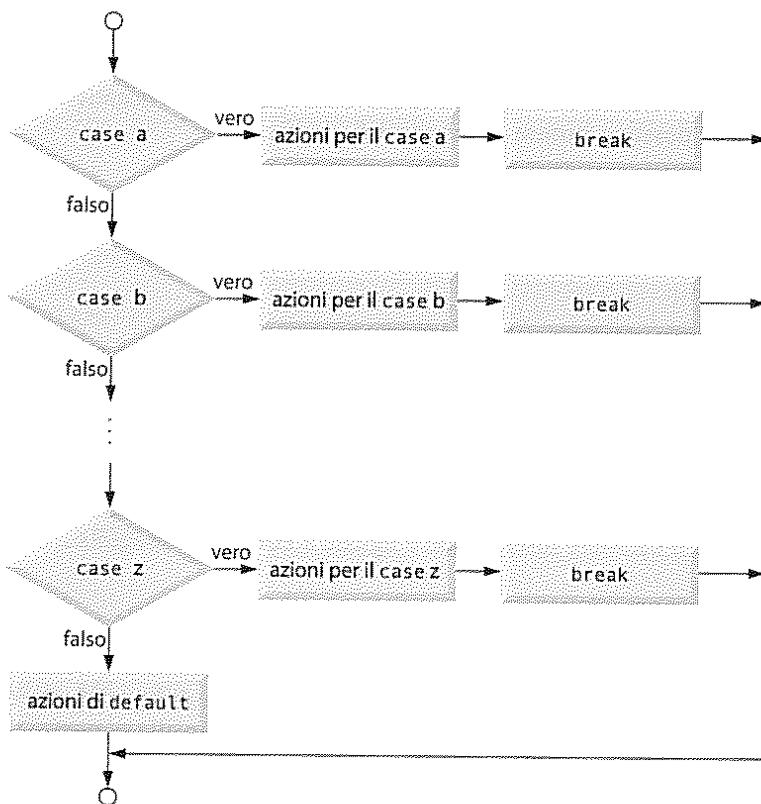
Se non si verificano confronti positivi, viene eseguito il caso `default`, che in questo programma stampa un messaggio di errore. Dovete sempre inserire un caso `default`; altrimenti, i valori non esplicitamente testati in uno `switch` saranno ignorati. Il caso `default` aiuta a evitare che ciò accada facendo focalizzare la vostra attenzione sulla necessità di processare condizioni eccezionali. Talvolta le elaborazioni di `default` non sono necessarie.

Benché le clausole `case` e la clausola del caso `default` in un'istruzione `switch` possano presentarsi in un ordine qualsiasi, è uso comune porre la clausola di `default` per ultima. Quando la clausola di `default` è

ultima, l'istruzione `break` finale non è necessaria. Molti programmatori includono comunque questo `break` per chiarezza e simmetria con gli altri casi.

Diagramma di flusso dell'istruzione switch

Il seguente diagramma di flusso, che rappresenta l'istruzione di selezione multipla `switch`, dimostra chiaramente che ogni istruzione `break` di un `case` provoca l'uscita immediata dall'istruzione `switch`.



Ignorare i caratteri di newline, tabulazione e spazio nell'input

Nell'istruzione `switch` della Figura 4.5, le righe

```

case '\n': // ignora i newline,
case '\t': // le tabulazioni
case ' ': // e gli spazi in input
    break;
    
```

fanno sì che il programma salti i caratteri di newline, tabulazione e spazio. La lettura dei caratteri uno alla volta può causare alcuni problemi. Per far sì che un programma legga i caratteri dovete inviarli al computer premendo *Invio*. Questo fa sì che venga inserito nell'input il carattere newline dopo il carattere che desideriamo elaborare.

Spesso, questo carattere newline (e altri caratteri di spazio vuoto) deve essere opportunamente ignorato per far funzionare il programma in modo corretto. I `case` precedenti nella nostra istruzione `switch` impediscono che il messaggio di errore nel caso `default` sia stampato ogni volta che nell'input si incontra un newline, una tabulazione o uno spazio. Ciascun input di questo esempio causa due iterazioni del ciclo: la prima per il voto a lettera e la seconda per '\n'. Scrivere diverse etichette `case` senza istruzioni intermedie significa che le stesse azioni vengono eseguite per ciascuno dei `case`.

Espressioni costanti integrali

Quando usate l'istruzione `switch`, ricordate che ogni singolo `case` può testare solamente un'**espressione costante integrale**, cioè qualsiasi combinazione di costanti di tipo carattere e di costanti intere il cui valore è un valore costante intero. Una costante di tipo carattere può essere rappresentata come un carattere specifico tra virgolette singole, come per esempio '`'A'`'. I caratteri devono essere racchiusi entro virgolette singole per essere riconosciuti come costanti di tipo carattere (i caratteri tra virgolette doppie sono riconosciuti come stringhe). Le costanti intere sono semplicemente valori interi. Nel nostro esempio abbiamo usato costanti di tipo carattere.

Note sui tipi interi

I linguaggi portabili come il C devono prevedere dimensioni flessibili per i tipi di dati. Le applicazioni possono richiedere interi di varie dimensioni. Il C fornisce diversi tipi di dati per rappresentare interi. Oltre a `int` e `char`, il C fornisce i tipi `short int` (che si può abbreviare in `short`) e `long int` (che si può abbreviare in `long`). Ci sono anche variazioni `unsigned` di tutti i tipi integrali che rappresentano valori interi non negativi. Nel Paragrafo 5.14, vedremo che il C fornisce anche il tipo `long long int` (che può essere abbreviato in `long long`).

Il C standard specifica l'intervallo minimo di valori per ogni tipo di intero, ma l'intervallo reale può essere maggiore, a seconda dell'implementazione. Per gli `short int`, l'intervallo minimo va da -32767 a +32767. Per la maggior parte dei calcoli con interi, i `long int` sono sufficienti. L'intervallo minimo di valori per i `long int` va da -2147483647 a +2147483647. L'intervallo di valori per un `int` è maggiore o uguale a quello di uno `short int` e minore o uguale a quello di un `long int`. Su molte delle piattaforme odierne gli `int` e i `long int` rappresentano lo stesso intervallo di valori. Il tipo di dati `signed char` può rappresentare interi nell'intervallo da -127 a +127 o uno qualunque dell'insieme di caratteri ASCII. Consultate la sezione 5.2.4.2 del documento del C standard per l'elenco completo degli intervalli minimi per i tipi interi `signed` e `unsigned`.

✓ Autovalutazione

- (Completare)** Talvolta, un algoritmo può contenere una serie di decisioni in cui una variabile o un'espressione è verificata separatamente per ognuno dei valori costanti interi che può assumere e per i quali vengono intraprese azioni differenti. Tale situazione è chiamata _____.

Risposta: selezione multipla.

- (Scelta multipla)** Quale delle seguenti affermazioni a), b) o c) è falsa?

- a) Il valore di un'assegnazione è il valore assegnato alla variabile sul lato sinistro dell'operatore =.
- b) Il valore dell'espressione di assegnazione `grade = getchar()` è il carattere restituito da `getchar` e assegnato alla variabile `grade`.
- c) L'istruzione seguente imposta a 0 le variabili a, b e c:

`0 = a = b = c;`

- d) Tutte le affermazioni precedenti sono vere.

Risposta: c) è falsa. L'istruzione corretta è:

`a = b = c = 0;`

4.7 Istruzione di iterazione do...while

L'istruzione di iterazione `do...while` è simile all'istruzione `while`. L'istruzione `while` verifica la sua condizione di continuazione del ciclo prima di eseguire il corpo del ciclo. L'istruzione `do...while` verifica la sua condizione di continuazione del ciclo dopo aver eseguito il corpo del ciclo, pertanto il corpo del ciclo sarà sempre eseguito almeno una volta. Quando un `do...while` termina, l'esecuzione prosegue con l'istruzione dopo la clausola `while`. La Figura 4.6 usa un'istruzione `do...while` per stampare i numeri da 1 a 5. Abbiamo deciso di preincrementare la variabile di controllo `counter` nel test di continuazione del ciclo (riga 10).

```

1 // fig04_06.c
2 // Uso dell'istruzione di iterazione do...while.
3 #include <stdio.h>
4
5 int main(void) {
6     int counter = 1; // inizializza il contatore
7
8     do {
9         printf("%d ", counter);
10    } while (++counter <= 5);
11 }

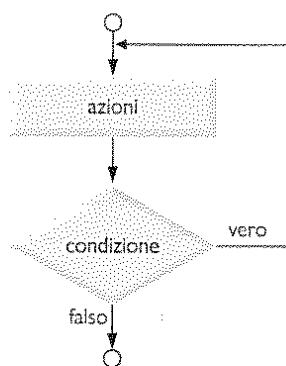
```

1 2 3 4 5

Figura 4.6 Uso dell'istruzione di iterazione do...while.

Diagramma di flusso dell'istruzione do...while

Il seguente diagramma di flusso dell'istruzione do...while rende evidente che la condizione di continuazione del ciclo non viene testata finché non viene eseguita per la prima volta l'azione del ciclo.



✓ Autovalutazione

- (Scelta multipla)** Quale delle seguenti affermazioni a), b) o c) è *falsa*?
 - L'istruzione while verifica la sua condizione di continuazione del ciclo prima di eseguire il corpo del ciclo, pertanto il corpo del ciclo sarà sempre eseguito almeno una volta.
 - L'istruzione do...while verifica la sua condizione di continuazione del ciclo dopo aver eseguito il corpo del ciclo.
 - Quando un do...while termina, l'esecuzione prosegue con l'istruzione dopo la clausola while.
 - Tutte le affermazioni precedenti sono *vere*.

Risposta: a) è *falsa*. In realtà, se la condizione di continuazione del ciclo dell'istruzione while risulta falsa quando entra nel ciclo, il corpo del ciclo non verrà eseguito.

- (Vero/Falso)** Assumendo che il contatore sia inizializzato a 1, il ciclo seguente stampa i numeri da 1 a 10:

```

do {
    printf("%d ", counter);
} while (++counter < 10);

```

Risposta: *Falso*. Questo ciclo stampa i numeri da 1 a 9. Per stampare i numeri da 1 a 10, si deve modificare < in <= nella condizione di continuazione del ciclo.

4.8 Istruzioni break e continue

Le istruzioni `break` e `continue` si usano per alterare il flusso di controllo. Il Paragrafo 4.6 ha mostrato come utilizzare l'istruzione `break` per terminare l'esecuzione di un'istruzione `switch`. Questo paragrafo esamina come usare `break` in un'istruzione di iterazione.

Istruzione break

L'istruzione `break`, quando è eseguita in un'istruzione `while`, `for`, `do...while` o `switch`, provoca un'uscita immediata da quell'istruzione. L'esecuzione del programma continua con l'istruzione successiva dopo quel `while`, `for`, `do...while` o `switch`. Normalmente, `break` serve per uscire subito da un ciclo o per saltare il resto di un'istruzione `switch` (come nella Figura 4.5). Il programma della Figura 4.7 mostra l'istruzione `break` (riga 12) in un'istruzione di iterazione `for`.

```

1 // fig04_07.c
2 // Uso dell'istruzione break in un'istruzione for.
3 #include <stdio.h>
4
5 int main(void) {
6     int x = 1; // dichiarato qui per un utilizzo dopo il ciclo
7
8     // ripeti 10 volte
9     for (; x <= 10; ++x) {
10         // se x e' 5, termina il ciclo
11         if (x == 5) {
12             break; // interrompi il ciclo solo se x e' 5
13         }
14
15         printf("%d ", x);
16     }
17
18     printf("\nBroke out of loop at x == %d\n", x);
19 }
```

```

1 2 3 4.
Broke out of loop at x == 5
```

Figura 4.7 Uso dell'istruzione `break` in un'istruzione `for`.

Quando l'istruzione `if` scopre che `x` vale 5, viene eseguito il `break`. Questo termina l'istruzione `for`, e il programma continua con il `printf` dopo il `for`. Il ciclo viene eseguito completamente solo quattro volte. Ricordatevi che quando dichiarate la variabile di controllo in un'espressione di *inizializzazione* di un ciclo `for`, la variabile non esiste più dopo la fine del ciclo. Abbiamo dichiarato e inizializzato `x` prima del ciclo in questo esempio, cosicché potremmo usare il suo valore finale dopo la fine del ciclo. Quindi, la sezione di *inizializzazione* dell'intestazione del `for` (prima del primo punto e virgola) è vuota.

Istruzione continue

L'istruzione `continue`, quando è eseguita in un'istruzione `while`, `for` o `do...while`, salta le istruzioni rimanenti nel corpo di quell'istruzione di controllo e fa eseguire la successiva iterazione del ciclo. Nelle istruzioni `while` e `do...while`, il test di continuazione del ciclo è valutato immediatamente dopo l'esecuzione dell'istruzione `continue`. Nell'istruzione `for` viene eseguita l'espressione di incremento, quindi viene valutato il test di continuazione del ciclo. Il programma della Figura 4.8 usa `continue` (riga 10) nell'istruzione `for` per saltare l'istruzione `printf` quando `x` è 5 e iniziare l'iterazione successiva del ciclo.

```

1 // fig04_08.c
2 // Uso dell'istruzione continue in un'istruzione for.
3 #include <stdio.h>
4
5 int main(void) {
6     // ripeti 10 volte
7     for (int x = 1; x <= 10; ++x) {
8         // se x e' 5, continua con la successiva iterazione del ciclo
9         if (x == 5) {
10             continue; // salta il restante codice nel corpo del ciclo
11         }
12
13         printf("%d ", x);
14     }
15
16     puts("\nUsed continue to skip printing the value 5");
17 }

```

1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5

Figura 4.8 Uso dell'istruzione continue in un'istruzione for.

Note su break e continue

Alcuni programmatore pensano che break e continue violino le norme di programmazione, quindi non le utilizzano. Gli effetti di queste istruzioni si possono ottenere per mezzo delle tecniche di programmazione strutturata che presto apprenderemo, ma le istruzioni break e continue sono eseguite più velocemente.

Vi è un contrasto tra l'obiettivo di realizzare un'ingegneria del software di qualità e quello di realizzare un software con le migliori prestazioni, e uno di questi due obiettivi viene raggiunto a spese dell'altro. In tutte le situazioni che non richiedono prestazioni spinte, osservate le seguenti linee guida: primo, rendete il vostro codice semplice e corretto; poi rendetelo veloce e compatto, ma solo se necessario.

✓ Autovalutazione

- (Scelta multipla) Quale delle seguenti affermazioni a), b) o c) è falsa?
 - L'istruzione break termina l'esecuzione di un'istruzione switch.
 - L'istruzione break, quando è eseguita in un'istruzione while, for o do...while, provoca un'uscita immediata da quell'istruzione.
 - L'uso di break serve normalmente per uscire subito da un ciclo o per saltare il resto di un'istruzione if...else.
 - Tutte le affermazioni precedenti sono vere.

Risposta: c) è falsa. In realtà, l'istruzione break salta il resto di un'istruzione switch, non di una if...else.

- (Scelta multipla) Quale delle seguenti affermazioni a), b) o c) è falsa?
 - L'istruzione continue, quando è eseguita in un'istruzione while, for o do...while, salta le istruzioni rimanenti nel corpo di quell'istruzione di controllo e fa eseguire la successiva iterazione del ciclo.
 - Nelle istruzioni while e do...while, il test di continuazione del ciclo è valutato immediatamente dopo l'esecuzione dell'istruzione continue.
 - Nell'istruzione for, dopo l'esecuzione dell'istruzione continue, viene valutato il test di continuazione del ciclo e quindi viene eseguita l'espressione di incremento.
 - Tutte le affermazioni precedenti sono vere.

Risposta: c) è *falsa*. In realtà, nell'istruzione `for`, dopo l'esecuzione dell'istruzione `continue`, viene eseguita l'espressione di incremento, quindi viene valutato il test di continuazione del ciclo.

4.9 Operatori logici

Finora abbiamo usato soltanto condizioni semplici, come `counter <= 10`, `total > 1000` e `grade != -1`. Abbiamo espresso queste condizioni in termini di operatori relazionali (`>`, `<`, `>=` e `<=`) e di operatori di uguaglianza (`==` e `!=`). Ogni decisione verificava precisamente una sola condizione. Per verificare varie condizioni nel processo decisionale, dovevamo eseguire questi test in istruzioni separate o in istruzioni annidate `if` o `if...else`. Il C fornisce **operatori logici** che si possono usare per formare condizioni più complesse combinando condizioni semplici. Gli operatori logici sono `&&` (AND logico), `||` (OR logico) e `!` (NOT logico, chiamato anche negazione logica). Considereremo esempi di ognuno di questi operatori.

Operatore AND logico (`&&`)

Supponiamo di volerci assicurare che due condizioni siano entrambe vere prima di scegliere un certo percorso di esecuzione. In questo caso, possiamo usare l'operatore logico `&&` come segue:

```
if (gender == 1 && age >= 65) {
    ++seniorFemales;
}
```

Questa istruzione `if` contiene due condizioni semplici. La condizione `gender == 1` potrebbe essere valutata, per esempio, per determinare se una persona è di sesso femminile. La condizione `age >= 65` determina se una persona è un cittadino anziano. Le due condizioni semplici sono valutate prima perché `==` e `>=` hanno una precedenza più alta di `&&`. L'istruzione `if` quindi considera la condizione combinata `gender == 1 && age >= 65`, che è *vera* se e solo se entrambe le condizioni semplici sono *vere*. Infine, se questa condizione combinata è *vera*, allora la precedente istruzione `if` incrementa `seniorFemales` di 1. Se una delle due o entrambe le condizioni semplici sono *false*, il programma salta il corpo di `if` e procede all'istruzione successiva in sequenza.

La tabella seguente riepiloga l'**operatore `&&`**.

espressione1	espressione2	espressione1 && espressione2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

La tabella mostra tutte e quattro le possibili combinazioni dei valori zero (*falso*) e nonzero (*vero*) per `espressione1` e `espressione2`. Tali tabelle sono spesso chiamate **tabelle di verità**. Il C assegna a tutte le espressioni che includono operatori relazionali, operatori di uguaglianza e/o operatori logici il valore 0 o 1. Benché il C assegni a 1 un valore *vero*, accetta come *vero* qualsiasi valore diverso da zero.

Operatore OR logico (`||`)

Consideriamo adesso l'operatore logico `||` (OR logico). Supponiamo di volerci assicurare, a un certo punto di un programma, che di due condizioni una o entrambe siano *vere* prima di scegliere un certo percorso di esecuzione. In questo caso, usiamo l'operatore `||`, come nel seguente segmento di programma:

```
if (semesterAverage >= 90 || finalExam >= 90) {
    puts("Student grade is A");
}
```

Questa istruzione contiene due condizioni semplici. La condizione `semesterAverage >= 90` determina se lo studente meriti una "A" come voto per un rendimento eccellente in tutto il semestre. La condizione `finalExam >= 90` determina se lo studente meriti una "A" come voto per un'eccezionale prestazione all'esame finale. L'istruzione `if` considera quindi la condizione combinata e assegna allo studente una "A" se una o entrambe le condizioni semplici sono *vere*. Viene stampato il messaggio "Student grade is A" a meno che entrambe le condizioni siano *false* (zero). La seguente è una tabella di verità per l'operatore OR logico (`||`).

espressione1	espressione2	espressione1 espressione2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Valutazione cortocircuitata

L'operatore `&&` ha una precedenza più alta di `||`. Entrambi gli operatori sono associativi da sinistra a destra. Un'espressione contenente gli operatori `&&` o `||` viene calcolata solo fino a quando non è noto se la condizione è *vera* o *falsa*. Quindi, la condizione

```
gender == 1 && age >= 65
```

si arresterà se `gender` non è uguale a 1 – (cioè se l'intera espressione è *falsa*) e continuerà se `gender` è uguale a 1 (l'intera espressione potrebbe essere *vera* se `age` è maggiore di 0 uguale a 65). Questa caratteristica ri-

- guardante le prestazioni per la valutazione delle espressioni logiche AND e OR è chiamata **valutazione cortocircuitata**.

Nelle espressioni che usano l'operatore `&&`, fate in modo che la condizione con maggiori probabilità di essere *falsa* sia quella più a sinistra. Nelle espressioni che usano l'operatore `||`, fate in modo che la condizione con maggiori probabilità di essere *vera* sia quella più a sinistra. Questo può ridurre il tempo di esecuzione di un programma.

Operatore NOT logico (!)

Il C fornisce l'operatore unario `!` (negazione logica) per permettervi di "invertire" il significato di una condizione. L'operatore logico di negazione ha soltanto una condizione singola come operando e lo si utilizza quando si è interessati a scegliere un percorso di esecuzione se la condizione dell'operando è *falsa*, come nel seguente segmento di programma:

```
if (!(grade == sentinelValue)) {
    printf("The next grade is %f\n", grade);
}
```

Le parentesi attorno alla condizione `grade == sentinelValue` sono necessarie perché l'operatore logico di negazione ha una precedenza più alta dell'operatore di uguaglianza. La seguente è una tabella di verità per l'operatore logico di negazione.

espressione	!espressione
0	1
nonzero	0

Nella maggior parte dei casi potete evitare di usare la negazione logica, esprimendo la condizione in modo diverso. Per esempio, l'istruzione precedente si può anche scrivere come segue:

```
if (grade != sentinelValue) {
    printf("The next grade is %f\n", grade);
}
```

Riepilogo della precedenza e dell'associatività degli operatori

La seguente tabella mostra la precedenza e l'associatività degli operatori introdotti fino a questo punto. Gli operatori sono mostrati dall'alto in basso in ordine decrescente di precedenza.

Operatori	Associatività	Tipo
<code>++ (postfisso)</code> <code>-- (postfisso)</code>	da destra a sinistra	postfisso
<code>+ - !</code> <code>++ (prefisso)</code> <code>-- (prefisso)</code> <code>(tipo)</code>	da destra a sinistra	unario
<code>*</code> <code>/</code> <code>%</code>	da sinistra a destra	moltiplicativo
<code>+</code> <code>-</code>	da sinistra a destra	additivo
<code><</code> <code><=</code> <code>></code> <code>>=</code>	da sinistra a destra	relazionale
<code>==</code> <code>!=</code>	da sinistra a destra	di uguaglianza
<code>&&</code>	da sinistra a destra	AND logico
<code> </code>	da sinistra a destra	OR logico
<code>? :</code>	da destra a sinistra	condizionale
<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code>	da destra a sinistra	di assegnazione
<code>,</code>	da sinistra a destra	virgola

Il tipo di dati `_Bool`

Il C standard comprende un tipo booleano – rappresentato dalla parola chiave `_Bool` – che può assumere solo i valori 0 o 1. Ricordatevi che il valore 0 in una condizione viene valutato come *falso*, mentre un valore diverso da zero viene valutato come *vero*. Assegnando un valore diverso da zero a un `_Bool`, questo viene impostato a 1. Lo standard include anche l'intestazione `<stdbool.h>`, che definisce `bool` come un'abbreviazione per il tipo `_Bool`, `true` e `false` come nomi simbolici di 1 e 0, rispettivamente. Al momento della preelaborazione, gli identificatori `bool`, `true` e `false` sono sostituiti con `_Bool`, 1 e 0, rispettivamente.

✓ Autovalutazione

1. (*Scelta multipla*) Quando viene eseguita la seguente istruzione `if`, quale coppia di valori delle variabili causa l'incremento del contatore `seniorFemales`?

```
if (gender == 1 && age >= 65) {
    ++seniorFemales;
}
```

- a) gender è 2 e age è 60.
- b) gender è 2 e age è 73.
- c) gender è 1 e age è 19.
- d) gender è 1 e age è 65.

Risposta: d.

2. (*Scelta multipla*) Quando viene eseguita la seguente istruzione `if`, quale coppia di valori delle variabili *non* determina la stampa del messaggio "Student grade is A"?

```
if (semesterAverage >= 90 || finalExam >= 90) {
    puts("Student grade is A");
}
```

- a) semesterAverage è 75 e finalExam è 80.
- b) semesterAverage è 85 e finalExam è 91.
- c) semesterAverage è 93 e finalExam è 67.
- d) semesterAverage è 94 e finalExam è 90.

Risposta: a.

⊗ 4.10 Confondere gli operatori di uguaglianza (==) e di assegnazione (=)

C'è un tipo di errore che i programmati in C, qualunque sia il loro livello di esperienza, tendono a commettere frequentemente e che merita un paragrafo a parte. Questo errore consiste nello scambiare accidentalmente tra loro gli operatori `==` (uguaglianza) e `=` (assegnazione). Ciò che rende questi scambi così dannosi è il fatto che essi, normalmente, non provocano errori di compilazione. Anzi, le istruzioni con questi errori vengono compilate in generale in modo corretto, permettendo ai programmi di essere eseguiti fino al completamento generando probabilmente risultati scorretti dovuti a errori logici al momento dell'esecuzione.

A causare questi problemi sono due aspetti del C. Uno consiste nel fatto che qualsiasi espressione che produce un valore può essere usata nella sezione di decisione di qualsiasi istruzione di controllo. Se il valore è 0 è trattata come *falsa*, se il valore è diverso da zero è trattata come *vera*. Il secondo consiste nel fatto che un'assegnazione ha un valore, ossia il valore assegnato alla variabile che compare sul lato sinistro dell'operatore `=`.

Per esempio, supponiamo di voler scrivere

```
if (payCode == 4) {
    printf("%s", "You get a bonus!");
}
```

ma accidentalmente scriviamo

```
if (payCode = 4) {
    printf("%s", "You get a bonus!");
}
```

La prima istruzione `if` in modo appropriato assegna un bonus alla persona il cui paycode (codice di pagamento) è uguale 4. La seconda istruzione `if` (quella con l'errore) calcola l'espressione di assegnazione nella condizione `if`. Dopo l'assegnazione, il valore nella condizione è 4. Dal momento che un qualunque valore diverso da zero è interpretato come *vera*, la condizione in questa istruzione `if` è sempre *vera*. Non solo il valore di `payCode` è inavvertitamente impostato a 4, ma la persona riceve sempre un bonus a prescindere da quale sia il valore del suo effettivo `payCode!` Usare inavvertitamente l'operatore `==` per l'assegnazione e usare inavvertitamente l'operatore `=` per l'uguaglianza sono entrambi errori logici.

Ivalue e rvalue

Probabilmente sarete propensi a scrivere condizioni come `x == 7` con il nome della variabile a sinistra e la costante a destra. Invertendo questi termini mettendo la costante a sinistra e il nome della variabile a destra, come in `7 == x`, se accidentalmente sostituite l'operatore `==` con `=`, sarete protetti dal compilatore. Il compilatore tratterà questo come un errore di sintassi poiché soltanto il nome di una variabile può essere posto alla sinistra di un'espressione di assegnazione. Questo eviterà l'effetto devastante di un errore logico al momento dell'esecuzione.

I nomi delle variabili sono detti *lvalue* (“left value” ovvero “valore sinistro”) perché si possono usare alla sinistra di un operatore di assegnazione. Le costanti sono dette *rvalue* (“right value” ovvero “valore destro”) perché si possono usare solo alla destra di un operatore di assegnazione. Un *lvalue* può essere usato anche come *rvalue*, ma non viceversa.

Confondere == e = in istruzioni semplici

L'altra faccia della medaglia può essere ugualmente spiacevole. Supponete di voler assegnare un valore a una variabile con un'istruzione semplice come

`x = 1;`

ma invece scrivete

`x == 1;`

- ⊗ Anche qui non si tratta di un errore di sintassi. Il compilatore valuta l'espressione condizionale. Se `x` è uguale a 1, la condizione è *vera* e l'espressione restituisce il valore 1. Se `x` non è uguale a 1, la condizione è *falsa* e l'espressione restituisce il valore 0. A prescindere da quale valore venga restituito, non vi è alcun operatore di assegnazione, così il valore viene semplicemente perso. Il valore di `x` rimane inalterato, provocando probabilmente un errore logico al momento dell'esecuzione. Sfortunatamente, non disponiamo di un trucco facile per aiutarvi in merito a questo problema! Molti compilatori, comunque, generano un avvertimento riguardo a una tale istruzione.

✓ Autovalutazione

1. (*Vero/Falso*) Lo scambio accidentale degli operatori `==` (uguaglianza) e `=` (assegnazione) è dannoso perché le istruzioni con questi errori vengono compilate in generale in modo corretto, permettendo ai programmi di essere eseguiti fino al completamento generando probabilmente risultati scorretti.

Risposta: *Vero*.

2. (*Vero/Falso*) Un *rvalue* può anche essere usato come *lvalue*, ma non viceversa.

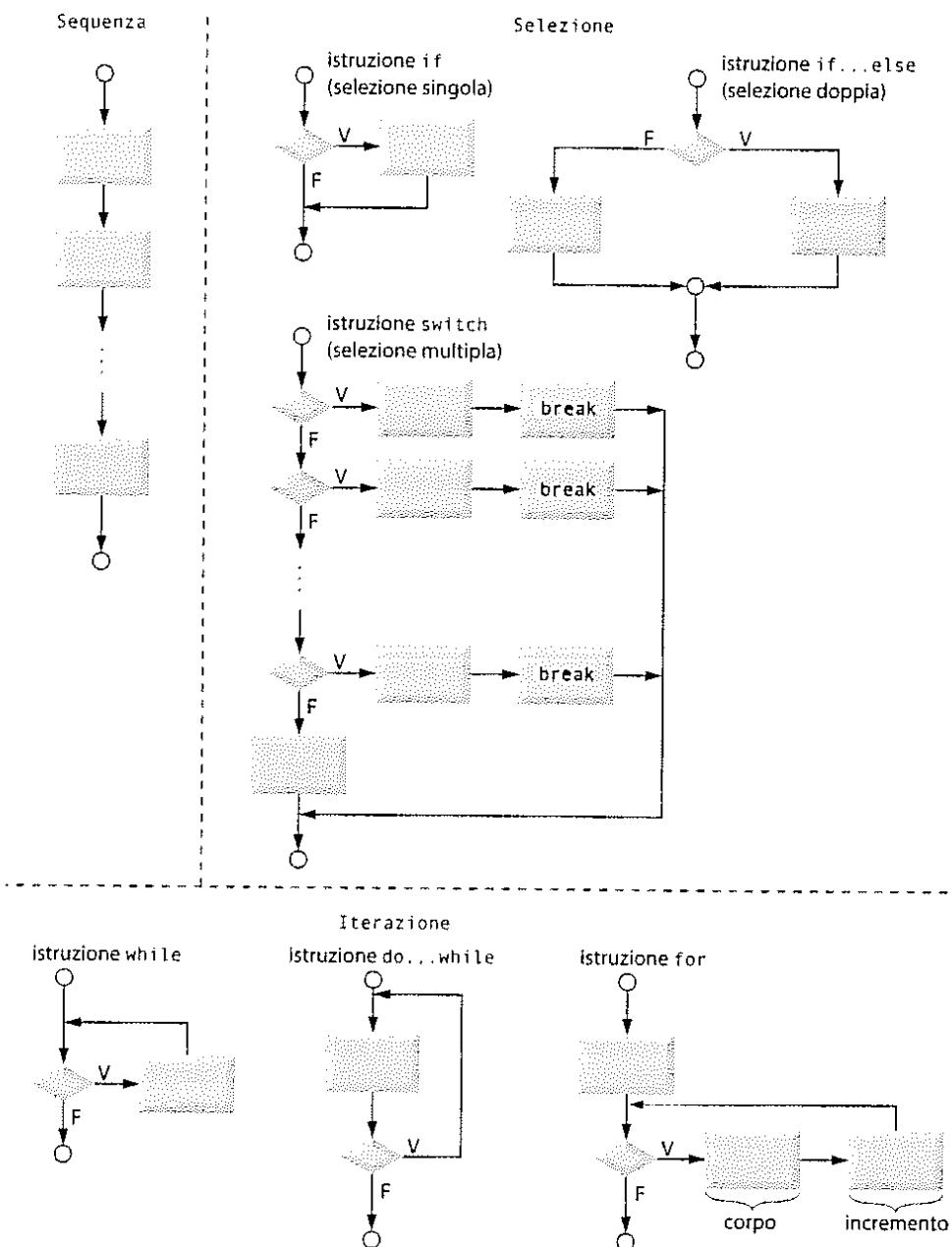
Risposta: *Falso*. In realtà, un *lvalue* può anche essere usato come *rvalue*, ma non viceversa.

4.11 Sintesi della programmazione strutturata

Proprio come gli architetti progettano costruzioni impiegando la sapienza collettiva della loro professione, nello stesso modo i programmati dovrebbero progettare i programmi. Il nostro campo è più giovane dell'architettura e la nostra sapienza collettiva è considerevolmente più scarsa. Abbiamo imparato moltissimo in appena novant'anni. Ma la cosa più importante, forse, è che abbiamo imparato che la programmazione strutturata produce programmi più facili (rispetto ai programmi non strutturati) da capire, verificare, correggere, modificare e persino dimostrare in senso matematico come corretti.

Nei Capitoli 3 e 4 sono state analizzate le istruzioni di controllo del C. Ora riassumiamo queste capacità e introduciamo un semplice insieme di regole per la costruzione di programmi strutturati. Il diagramma seguente riassume i diagrammi di flusso delle istruzioni di controllo.

Nel diagramma, i cerchietti indicano l'unico punto di entrata e l'unico punto di uscita di ciascuna istruzione. Collegare arbitrariamente simboli individuali dei diagrammi di flusso può portare a programmi non strutturati. Pertanto, la professione della programmazione ha scelto di combinare i simboli dei diagrammi di flusso in un insieme limitato di istruzioni di controllo e di costruire soltanto programmi strutturati opportunamente combinando correttamente le istruzioni di controllo in due semplici modi. Per semplicità sono usate unicamente istruzioni di controllo con un solo ingresso e una sola uscita, e potete combinarle soltanto impilandole in sequenza o annidandole.

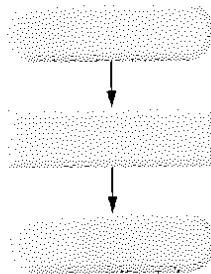


Regole per costruire programmi strutturati

La seguente tabella riepiloga le regole per costruire programmi strutturati. Partiamo dal presupposto che il simbolo rettangolo dei diagrammi di flusso possa essere usato per indicare qualsiasi azione, incluso l'input/output.

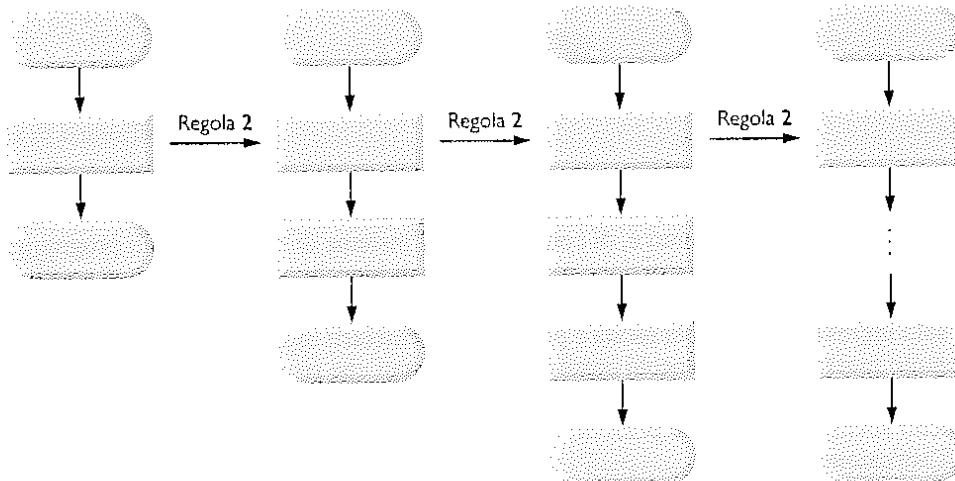
Regole per costruire programmi strutturati

1. Iniziare con il "diagramma di flusso più semplice" mostrato nel diagramma seguente.
2. Regola di "accatastamento": qualsiasi rettangolo (azione) può essere sostituito con due rettangoli (azioni) in sequenza.
3. Regola di "annidamento": qualsiasi rettangolo (azione) può essere sostituito con qualsiasi istruzione di controllo (sequenza, if, if...else, switch, while, do...while o for).
4. Le regole 2 e 3 possono essere applicate quante volte si vuole e in *qualsiasi* ordine.



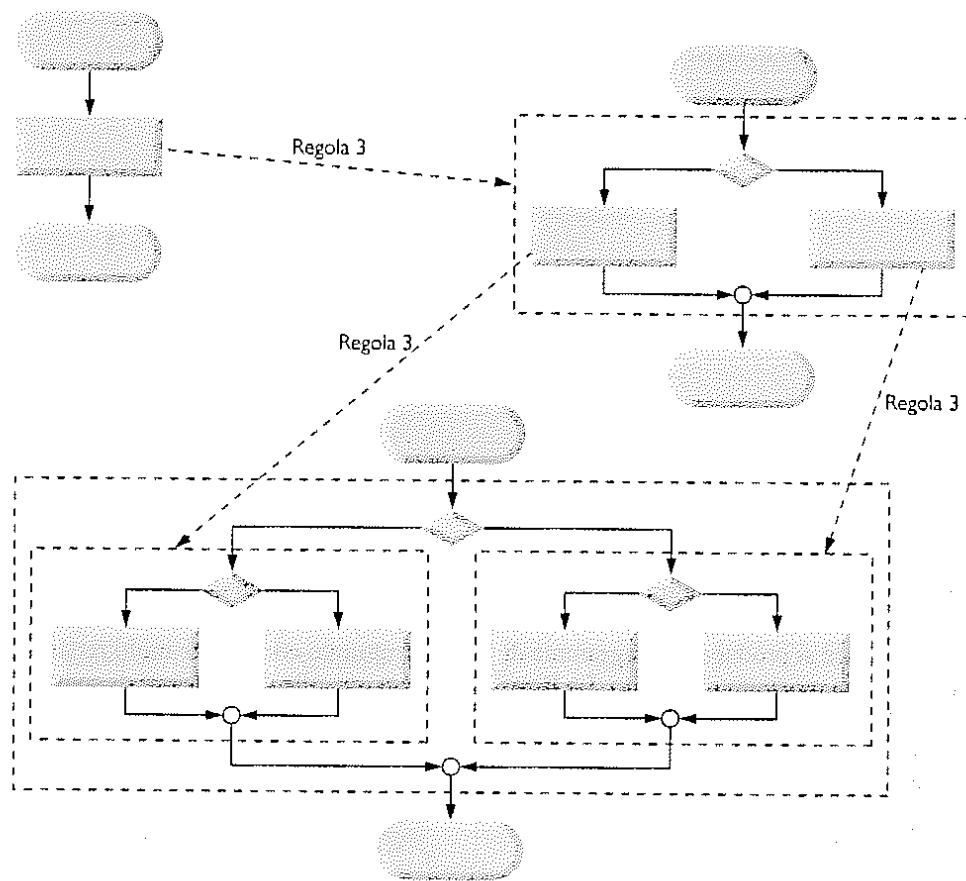
Regole per costruire programmi strutturati: regola di "accatastamento"

Applicando le regole per costruire programmi strutturati si produce sempre un diagramma di flusso strutturato che si presenta in modo chiaro come un blocco costituente. Applicando ripetutamente la Regola 2 al diagramma di flusso più semplice si produce un diagramma di flusso strutturato contenente molti rettangoli in sequenza, come nel diagramma mostrato di seguito. La Regola 2 genera una pila di istruzioni di controllo, e pertanto la chiamiamo **regola di accatastamento**.



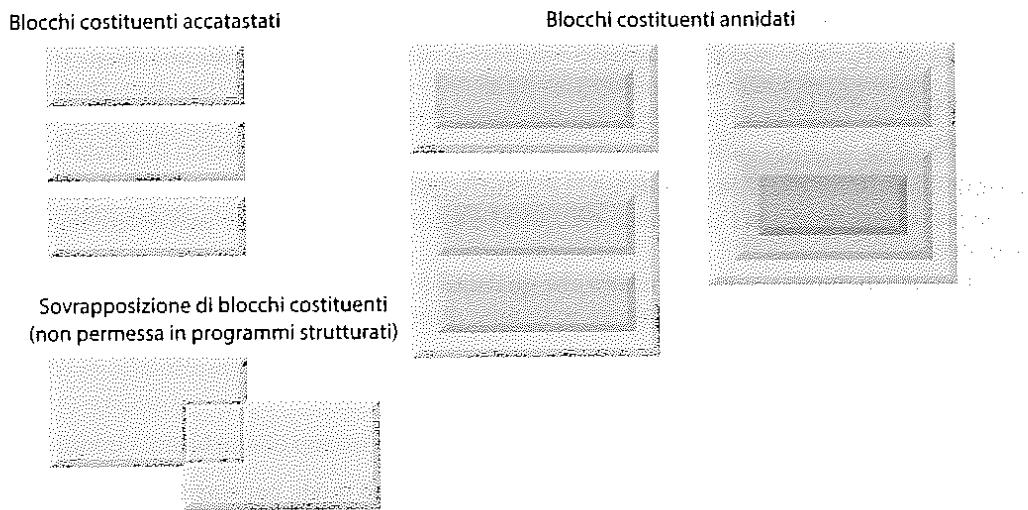
Regole per costruire programmi strutturati: regola di "annidamento"

La Regola 3 è chiamata **regola di annidamento**. Applicando ripetutamente la Regola 3 al diagramma di flusso più semplice si produce un diagramma di flusso con le istruzioni di controllo annidate in maniera evidente. Per esempio, nel diagramma mostrato di seguito, il rettangolo nel diagramma di flusso più semplice è sostituito con un'istruzione di selezione doppia (`if...else`). Dopodiché, si applica di nuovo la Regola 3 a entrambi i rettangoli nell'istruzione di selezione doppia, sostituendo ognuno di essi con istruzioni di selezione doppia. Il riquadro tratteggiato attorno a ognuna delle istruzioni di selezione doppia rappresenta il rettangolo che è stato sostituito nel diagramma di flusso originario.

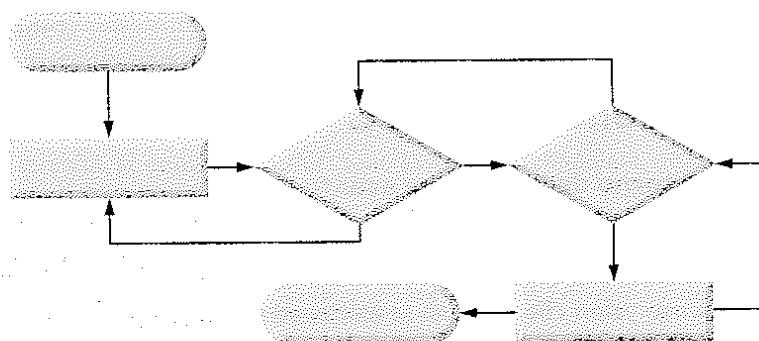


La Regola 4 genera strutture più ampie, più complicate e annidate più in profondità. I diagrammi di flusso che risultano dall'applicazione delle regole per la creazione di programmi strutturati costituiscono l'insieme di tutti i possibili diagrammi di flusso strutturati e, di conseguenza, l'insieme di tutti i possibili programmi strutturati.

È a causa dell'eliminazione dell'istruzione `goto` che questi blocchi costituenti non si sovrappongono mai l'uno sull'altro. La bellezza dell'approccio strutturato è che noi usiamo soltanto un piccolo numero di pezzi semplici a un solo ingresso e a una sola uscita e possiamo assemblarli in due soli semplici modi. Il diagramma mostrato di seguito mostra i tipi di blocchi costituenti accatastati che risultano dall'applicazione della Regola 2 e i tipi di blocchi annidati che risultano dall'applicazione della Regola 3. La figura mostra anche il tipo di blocchi costituenti sovrapposti che *non possono* comparire in diagrammi di flusso strutturati (a causa dell'eliminazione dell'istruzione `goto`).



Se si seguono le regole per la costruzione di programmi strutturati, *non si può* creare un diagramma di flusso non strutturato, come quello nel diagramma mostrato di seguito.



Se non siete certi che un particolare diagramma di flusso sia strutturato, applicate inversamente le regole per creare programmi strutturati per cercare di ridurre il diagramma di flusso al diagramma di flusso più semplice. Se ci riuscite, il diagramma originario è strutturato, altrimenti non lo è.

Tre forme di controllo

La programmazione strutturata favorisce la semplicità. Böhm e Jacopini hanno dimostrato che sono necessarie solo tre forme di controllo:

- sequenza;
- selezione;
- iterazione.

La sequenza è lineare. La selezione è implementata in uno dei tre modi:

- istruzione `if` (selezione singola);
- istruzione `if...else` (selezione doppia);
- istruzione `switch` (selezione multipla).

È facile dimostrare che la semplice istruzione `if` è sufficiente a realizzare qualunque forma di selezione, ossia tutto quello che si può fare con l'istruzione `if...else` e l'istruzione `switch` può essere implementato con una o più istruzioni `if`.

L'iterazione è implementata in uno dei seguenti tre modi:

- istruzione `while`;
- istruzione `do...while`;
- istruzione `for`.

È inoltre facile dimostrare che l'istruzione `while` è sufficiente per ottenere qualunque forma di iterazione. Tutto ciò che può essere fatto con l'istruzione `do...while` e con l'istruzione `for` può essere implementato con l'istruzione `while`.

La combinazione di questi risultati dimostra che qualsiasi forma di controllo che possa mai risultare necessaria in un programma in C può essere espressa in termini di *tre* sole forme di controllo:

- sequenza;
- istruzione `if` (selezione);
- istruzione `while` (iterazione).

E queste istruzioni di controllo possono essere combinate in *due* soli modi; *accatastamento* e *annidamento*. La programmazione strutturata favorisce davvero la semplicità.

Nei Capitoli 3 e 4 abbiamo esaminato come comporre programmi partendo da istruzioni di controllo contenenti solo azioni e decisioni. Nel Capitolo 5 introdurremo un'altra unità strutturale dei programmi, chiamata funzione. Impareremo a comporre programmi di grandi dimensioni combinando funzioni che, a loro volta, possono essere composte da istruzioni di controllo. Esamineremo anche come l'uso di funzioni favorisca la riusabilità del software.



4.12 Programmazione sicura in C

Controllare il valore di ritorno della funzione `scanf`

La Figura 4.4 ha usato la funzione `pow` della libreria `math`, che calcola il valore del suo primo argomento elevato alla potenza del suo secondo argomento e *restituisce* il risultato come valore `double`. Il risultato del calcolo è stato quindi impiegato nell'istruzione che chiamava la funzione `pow`.

Molte funzioni restituiscono valori che indicano se esse sono state eseguite con successo. Per esempio, la funzione `scanf` restituisce un `int` che indica se l'operazione di input è riuscita. Se si verifica un fallimento dell'input prima che `scanf` possa leggere un valore, `scanf` restituisce il valore `EOF` (definito in `<stdio.h>`); altrimenti, restituisce il numero di elementi che sono stati letti nelle variabili. Se questo valore *non* corrisponde al numero degli elementi che intendevate far leggere, significa che `scanf` non è stata in grado di completare l'operazione di input.

Considerate la seguente istruzione che si aspetta di leggere un valore `int` in `grade`:

```
scanf("%d", &grade); // legge il voto dell'utente
```

Se l'utente inserisce un numero intero, `scanf` restituisce 1, il quale indica che un valore è stato realmente letto. Se l'utente inserisce una stringa, come "hello", `scanf` restituisce 0, il quale indica che non è stata in grado di convertire l'input in un intero. In questo caso, la variabile `grade` *non* riceve alcun valore.

La funzione `scanf` può leggere più valori in ingresso, come in

```
scanf("%d%d", &number1, &number2); // legge due interi
```

Se l'input in entrambe le variabili riesce, `scanf` restituirà 2. Se l'utente inserisce una stringa per il primo valore, `scanf` restituirà 0, e non riceveranno valori né `number1` né `number2`. Se l'utente inserisce un intero seguito da una stringa, `scanf` restituirà 1 e solo `number1` riceverà un valore.

Per rendere più robusto il trattamento dell'input controllate il valore di ritorno di `scanf` per assicurarvi che il numero di valori letti corrisponda al numero di valori attesi. Altrimenti, il vostro programma userà i valori delle variabili come se `scanf` avesse completato l'elaborazione con successo. Questo potrebbe portare a errori logici, o anche a un arresto del programma o persino a vulnerabilità ad attacchi.



Controllo degli intervalli

Anche se `scanf` opera con successo, i valori letti potrebbero essere ancora non validi. Per esempio, i voti sono tipicamente numeri interi nell'intervallo da 0 a 100. In un programma che riceve in ingresso tali voti, dovreste **convalidare** ciascun voto usando il **controllo dell'intervallo** per assicurarvi che siano valori da 0 a 100. Potete poi chiedere all'utente di reinserire qualunque valore dato al di fuori dell'intervallo. Se un programma richiede che l'input faccia parte di uno specifico insieme di valori (es. codici non sequenziali di prodotti), dovreste controllare che ogni valore in input corrisponda a un valore dell'insieme.³

✓ Autovalutazione

- (Completare)** Se si verifica un fallimento dell'input, `scanf` restituisce il valore _____; altrimenti, restituisce il numero di elementi che sono stati letti.

Risposta: EOF.

- (Scelta multipla)** Considerando questo codice:

```
scanf ("%d%d", &grade1, &grade2); // leggi due interi
```

quale delle seguenti affermazioni a), b) o c) è falsa?

- a) Se l'input riesce, `scanf` restituirà 0, il quale indica che sono stati inseriti interi come valori per entrambe le variabili.
- b) Se l'utente inserisce una stringa come primo valore, `scanf` restituirà 0, e non riceveranno valori né `grade1` né `grade2`.
- c) Se l'utente inserisce un intero seguito da una stringa, `scanf` restituirà 1, e solo `grade1` riceverà un valore.
- d) Tutte le affermazioni precedenti sono vere.

Risposta: a) è falsa. In realtà, se entrambi i valori sono inseriti correttamente, `scanf` restituirà 2.

4.13 Riepilogo

Paragrafo 4.2 Aspetti essenziali dell'iterazione

- La maggior parte dei programmi ha al suo interno il costrutto di iterazione, ovvero un ciclo. Un ciclo è un gruppo di istruzioni che il computer esegue ripetutamente finché una qualche **condizione di continuazione del ciclo** rimane vera.
- L'iterazione controllata da contatore usa una **variabile di controllo** per contare il numero di iterazioni. Quando è stato eseguito il numero corretto di iterazioni, il ciclo termina e il programma riprende l'esecuzione con l'istruzione dopo l'istruzione di iterazione.
- Nell'iterazione controllata da sentinella, un **valore sentinella** viene inserito dopo tutti i dati regolari per indicare la "fine dei dati". Le sentinelle devono essere distinte dai dati regolari.

Paragrafo 4.3 Iterazione controllata da contatore

- L'iterazione controllata da contatore necessita del **nome** della variabile di controllo, del suo **valore iniziale**, dell'**incremento** (o **decremento**) con il quale essa viene modificata ogni volta nel corso del ciclo, e della condizione che verifica il **valore finale** della variabile di controllo.
- La variabile di controllo è **incrementata** (o **decrementata**) a ogni esecuzione del gruppo di istruzioni.

Paragrafo 4.4 Istruzione di iterazione for

- L'**istruzione di iterazione for** gestisce tutti i dettagli dell'iterazione controllata da contatore.

3. Per ulteriori informazioni, consultate il Capitolo 5, "Integer Security", del libro *Secure Coding in C and C++*, 2/e di Robert Seacord.

- Quando un'istruzione `for` inizia l'esecuzione, la sua variabile di controllo viene inizializzata. Viene quindi controllata la condizione di continuazione del ciclo. Se la condizione è vera, viene eseguito il corpo del ciclo. La variabile di controllo è allora incrementata, e la condizione di continuazione del ciclo viene verificata. Questo processo continua finché la condizione di continuazione del ciclo non diventa falsa.
- Il formato generale dell'istruzione `for` è

```
for (inizializzazione; condizione; incremento) {
    istruzioni
}
```

dove l'espressione *inizializzazione* inizializza (e forse definisce) la variabile di controllo, l'espressione *condizione* è la condizione di continuazione del ciclo e l'espressione *incremento* incrementa la variabile di controllo.

- Le tre espressioni nell'intestazione `for` sono opzionali. Se la *condizione* è omessa, il C suppone che la condizione sia vera, generando così un ciclo infinito. Si può omettere l'espressione *inizializzazione* se la variabile di controllo è inizializzata prima del ciclo. Si può omettere l'espressione *incremento* se l'incremento è calcolato da istruzioni nel corpo dell'istruzione `for` o se non è necessario alcun incremento.
- L'intestazione `for` richiede l'uso di due punti e virgola.
- L'"incremento" può essere negativo per creare un ciclo che conti verso il basso.
- Se la condizione di continuazione del ciclo è inizialmente falsa, il corpo del ciclo non viene eseguito. Invece, l'esecuzione procede con l'istruzione che segue l'istruzione `for`.

Paragrafo 4.5 Esempi di uso dell'istruzione `for`

- La funzione `pow` effettua l'esponenziazione. La chiamata `pow(x, y)` calcola il valore di `x` elevato alla potenza `y`. La funzione riceve due argomenti `double` e restituisce un `double`.
- L'intestazione `<math.h>` deve essere inclusa ogni volta che si usa una funzione matematica come `pow`.
- La specifica di conversione `%21.2f` indica che un valore in virgola mobile sarà stampato **allineato a destra** in un campo di 21 caratteri con due cifre alla destra del punto decimale.
- Per **allineare a sinistra** un valore in un campo, mettete un - (segno meno) tra il simbolo % e la larghezza di campo.

Paragrafo 4.6 Istruzione di selezione multipla `switch`

- Talvolta, un algoritmo conterrà una serie di decisioni in cui una variabile o un'espressione è verificata separatamente per ognuno dei valori interi costanti che può assumere, e per ognuno di loro sono intraprese azioni differenti. Questa è chiamata **selezione multipla** e per gestirla il C fornisce l'istruzione `switch`.
- I caratteri sono normalmente memorizzati in variabili di tipo `char`. I caratteri possono essere memorizzati in qualsiasi tipo di dati interi, perché nel computer sono solitamente rappresentati come interi di un byte. In questo modo, possiamo trattare un carattere o come un intero o come un carattere, a seconda del suo uso.
- L'istruzione `switch` consiste in una serie di etichette `case`, un caso `default` opzionale e le istruzioni da eseguire per ogni `case`.
- La funzione `getchar` (intestazione `<stdio.h>`) legge e restituisce come `int` un carattere digitato sulla tastiera.
- Molti computer usano oggi l'insieme di caratteri Unicode. ASCII è un sottoinsieme di Unicode.
- I caratteri possono essere letti con `scanf` usando la specifica di conversione `%c`.
- Le **espressioni di assegnazione** nel loro insieme hanno esse stesse un **valore**, che è assegnato alla variabile alla sinistra dell'operatore `=`.
- `EOF` è spesso usato come valore sentinella. `EOF` è una costante intera simbolica definita in `<stdio.h>`.

- Sui sistemi macOS/Linux, l'indicatore EOF viene inserito digitando ***Ctrl + d***. Su Windows, l'indicatore EOF si può inserire digitando ***Ctrl + z***.
- La parola chiave **switch** è seguita dall'**espressione di controllo** tra parentesi. Il valore di questa espressione è confrontato con ognuna delle etichette **case**. Se il confronto ha successo, vengono eseguite le istruzioni per quel **case**. Se il confronto non ha successo, vengono eseguite le istruzioni del caso **default**.
- L'**istruzione break** fa continuare l'esecuzione del programma dall'istruzione dopo lo **switch**. L'istruzione **break** impedisce che i **case** di un'istruzione **switch** siano eseguiti tutti insieme.
- Ogni **case** può avere una o più azioni. Non sono richieste parentesi graffe attorno alle sequenze di azioni in un **case** di uno **switch**.
- **Mettere assieme diverse etichette case** fa sì che venga eseguito lo stesso insieme di azioni per ognuno di questi **case**.
- Ogni **case** in un'istruzione **switch** può testare solo una **espressione costante integrale**, cioè una qualunque combinazione di costanti di caratteri e costanti intere il cui valore calcolato sia un valore costante intero. Una costante di tipo carattere si può rappresentare come il carattere tipografico tra virgolette singole, come '**A**'. I caratteri devono essere chiusi tra virgolette singole per essere riconosciuti come costanti di tipo carattere. Le costanti intere sono semplicemente valori interi.
- Oltre ai tipi interi **int** e **char**, il C fornisce i tipi **short int** (che si può abbreviare con **short**) e **long int** (che si può abbreviare con **long**), così come versioni **unsigned** di tutti i tipi interi. Il C standard specifica l'intervallo minimo di valori per ogni tipo. L'intervallo reale può essere molto più grande e dipende dall'implementazione. Per gli **short int**, l'intervallo minimo è da -32767 a +32767. L'intervallo minimo di valori per i **long int** è da -2147483647 a +2147483647. L'intervallo di valori per un **int** è maggiore o uguale a quello di un **short int** e minore o uguale a quello di un **long int**. Su molte delle odierne piattaforme, gli **int** e i **long int** rappresentano lo stesso intervallo di valori. I tipi di dati **signed char** possono essere usati per rappresentare interi nell'intervallo da -127 a +127 o uno qualsiasi dei caratteri dell'insieme ASCII.

Paragrafo 4.7 Istruzione di iterazione do...while

- L'**istruzione do...while** verifica la condizione di continuazione del ciclo *dopo* l'esecuzione del corpo del ciclo. Pertanto, il corpo del ciclo viene eseguito almeno una volta. Quando un **do...while** termina, l'esecuzione continua con l'istruzione dopo la clausola **while**.

Paragrafo 4.8 Istruzioni break e continue

- L'**istruzione break**, quando è eseguita in un'istruzione **while**, **for**, **do...while** o **switch**, causa immediatamente l'uscita dall'istruzione. L'esecuzione del programma continua con l'istruzione successiva.
- L'**istruzione continue**, quando è eseguita in un'istruzione **while**, **for** o **do...while**, salta le restanti istruzioni nel corpo ed esegue l'iterazione successiva del ciclo. Le istruzioni **while** e **do...while** valutano il test di continuazione del ciclo immediatamente. Un'istruzione **for** esegue la sua espressione di incremento, poi testa la condizione di continuazione del ciclo.

Paragrafo 4.9 Operatori logici

- Gli **operatori logici** **&&** (AND logico), **||** (OR logico) e **!** (NOT logico, o negazione logica) possono essere usati per formare condizioni complesse combinando condizioni semplici.
- L'operatore **&& (AND logico)** restituisce *vero* se e solo se entrambi gli operandi sono *veri*.
- Il C valuta tutte le espressioni che includono operatori relazionali, operatori di uguaglianza e/o operatori logici. Sebbene il C assegna 1 a un **valore vero**, accetta come *vero* qualsiasi **valore diverso da zero**.
- L'operatore **|| (OR logico)** restituisce *vero* se uno o entrambi i suoi operandi sono *veri*.
- L'operatore **&&** ha una precedenza più alta di **||**. Entrambi gli operatori sono associativi da sinistra a destra.

- Gli operatori `&&` o `||` usano una valutazione cortocircuitata, che termina non appena diventa noto se la condizione è falsa o vera.
- Il C fornisce l'operatore `!` (**negazione logica**) per permettervi di “invertire” il significato di una condizione. Diversamente dagli operatori binari `&&` e `||`, che combinano due condizioni, l'operatore unario di negazione logica ha soltanto una singola condizione come operando.
- L'operatore di negazione logica è posto prima di una condizione quando siamo interessati a scegliere un percorso di esecuzione se la condizione originaria (senza l'operatore di negazione logica) è *falsa*.

Paragrafo 4.10 Confondere gli operatori di uguaglianza (`==`) e di assegnazione (`=`)

- I programmati spesso scambiano accidentalmente tra loro gli operatori `==` e `=`. Le istruzioni con questi errori di solito vengono compilate correttamente, permettendo l'esecuzione completa dei programmi, mentre verosimilmente generano risultati scorretti a causa di errori logici al momento dell'esecuzione.
- Se in una condizione come `7 == x` sostituite inavvertitamente l'operatore `==` con `=`, il compilatore segnalerà un errore di sintassi. Soltanto il nome di una variabile può essere posto alla sinistra di un'espressione di assegnazione.
- I nomi delle variabili sono detti *lvalue* (per “*left value*” ovvero “valore sinistro”) perché possono essere usati alla sinistra di un operatore di assegnazione.
- Le costanti sono dette *rvalue* (for “*right value*” ovvero “valore destro”) perché possono essere usate soltanto alla destra di un operatore di assegnazione. Gli *lvalue* possono anche essere usati come *rvalue*, ma non viceversa.

Esercizi di autovalutazione

- 4.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
 - a) Nell'iterazione controllata da contatore si usa una _____ per contare il numero di volte in cui si deve ripetere un gruppo di istruzioni.
 - b) L'istruzione _____, quando è eseguita in un'istruzione di iterazione, fa sì che sia eseguita immediatamente la successiva iterazione del ciclo.
 - c) L'istruzione _____, quando è eseguita in un'istruzione di iterazione o in uno `switch`, provoca un'uscita immediata dall'istruzione.
 - d) L'_____ è usata per testare una particolare variabile o espressione in relazione a ognuno dei valori costanti interi che può assumere.
- 4.2 Stabilite se le seguenti affermazioni sono *vere* o *false*. Se la risposta è *falsa*, spiegate perché.
 - a) Il caso `default` è necessario nell'istruzione di selezione `switch`.
 - b) L'istruzione `break` è necessaria nel caso `default` di un'istruzione `switch`.
 - c) L'espressione `(x > y && a < b)` è *vera* se `x > y` è *vera* o se `a < b` è *vera*.
 - d) Un'espressione contenente l'operatore `||` è *vera* se uno o entrambi gli operandi sono *veri*.
- 4.3 Scrivete un'istruzione o un insieme di istruzioni per eseguire ognuno dei seguenti compiti.
 - a) Sommate i numeri interi dispari tra 1 e 99 usando un'istruzione `for`. Utilizzate variabili intere `sum` e `count`.
 - b) Stampate il valore 333.546372 in una larghezza di campo di 15 caratteri con precisioni di 1, 2, 3, 4 e 5. Allineate a sinistra l'output. Quali sono i cinque valori che vengono stampati?
 - c) Calcolate il valore di 2.5 elevato alla potenza di 3 usando la funzione `pow`. Stampate il risultato con una precisione di 2 in una larghezza di campo di 10 posizioni. Qual è il valore che viene stampato?
 - d) Stampate gli interi da 1 a 20 usando un ciclo `while` e la variabile contatore `x`. Stampate soltanto cinque interi per riga. [Suggerimento: usate il calcolo `x % 5`. Quando il valore di questo è 0, stampate un carattere newline, altrimenti stampate un carattere di tabulazione.]
 - e) Ripetete l'Esercizio 4.3(d) usando un'istruzione `for`.

4.4 Trovate l'errore in ognuno dei seguenti segmenti di codice e spiegate come correggerlo:

- a)

```
x = 1;
while (x <= 10);
    ++x;
}
```
- b)

```
for (double y = .1; y != 1.0; y += .1) {
    printf("%f\n", y);
}
```
- c)

```
switch (n) {
    case 1:
        puts("The number is 1");
    case 2:
        puts("The number is 2");
        break;
    default:
        puts("The number is not 1 or 2");
        break;
}
```

- d) Il seguente codice deve stampare i valori da 1 a 10.

```
n = 1;
while (n < 10) {
    printf("%d ", n++);
}
```

Risposte agli esercizi di autovalutazione

- 4.1 a) variabile di controllo o contatore. b) continue. c) break. d) istruzione di selezione switch.
- 4.2 a) *Falso*. Il caso default è opzionale. Se non è necessaria alcuna azione di default, allora non c'è necessità di un caso default.
 b) *Falso*. L'istruzione break si usa per uscire dall'istruzione switch. L'istruzione break non è necessaria in *nessun* case.
 c) *Falso*. Entrambe le espressioni relazionali devono essere *vere* affinché l'intera espressione sia *vera* quando si usa l'operatore &&.
 d) *Vero*.
- 4.3 a)

```
int sum = 0;
for (int count = 1; count <= 99; count += 2) {
    sum += count;
}
```
- b)

```
printf("%-15.1f\n", 333.546372); // stampa 333.5
printf("%-15.2f\n", 333.546372); // stampa 333.55
printf("%-15.3f\n", 333.546372); // stampa 333.546
printf("%-15.4f\n", 333.546372); // stampa 333.5464
printf("%-15.5f\n", 333.546372); // stampa 333.54637
```
- c)

```
printf("%10.2f\n", pow(2.5, 3)); // stampa 15.63
```
- d)

```
int x = 1;
while (x <= 20) {
    printf("%d", x);
    if (x % 5 == 0) {
        puts("");
    }
    else {
```

```

        printf("%s", "\t");
    }
    ++x;
}

```

oppure

```

int x = 1;
while (x <= 20) {
    if (x % 5 == 0) {
        printf("%d\n", x++);
    }
    else {
        printf("%d\t", x++);
    }
}

```

oppure

```

int x = 0;
while (++x <= 20) {
    if (x % 5 == 0) {
        printf("%d\n", x);
    }
    else {
        printf("%d\t", x);
    }
}

```

e) `for (int x = 1; x <= 20; ++x) {`
 `printf("%d", x);`
 `if (x % 5 == 0) {`
 `puts("");`
 `}`
 `else {`
 `printf("%s", "\t");`
 `}`
`}`

oppure

```

for (int x = 1; x <= 20; ++x) {
    if (x % 5 == 0) {
        printf("%d\n", x);
    }
    else {
        printf("%d\t", x);
    }
}

```

- 4.4 a) Errore: il punto e virgola dopo l'intestazione del `while` causa un ciclo infinito.
Correzione: sostituite il punto e virgola con `{` oppure rimuovete sia il `;` che la `}{`.
b) Errore: viene usato un numero in virgola mobile per controllare un'istruzione di iterazione `for`.
Correzione: usare un intero ed eseguire le opportune operazioni per ottenere i valori desiderati.

```

for (int y = 1; y != 10; ++y) {
    printf("%f\n", (float) y / 10);
}

```

- c) Errore: manca l'istruzione `break` nelle istruzioni per il primo `case`.

Correzione: aggiungere un'istruzione `break` alla fine delle istruzioni per il primo `case`. Questo non è necessariamente un errore se volete che l'istruzione di `case 2:` venga eseguita ogni volta che viene eseguita l'istruzione di `case 1:`.

- d) Errore: viene usato un operatore relazionale non appropriato nella condizione di continuazione dell'iterazione del `while`.

Correzione: usate `<=` anziché `<`.

Esercizi

- 4.5 Trovate l'errore in ciascuno dei seguenti segmenti di codice. (*Nota:* potrebbe esserci più di un errore.)

a) `For (x = 100, x >= 1, ++x) {`
 `printf("%d\n", x);`
`}`

- b) Il seguente codice deve stampare un messaggio che dice se un dato numero intero è dispari o pari.

```

switch (value % 2) {
    case 0:
        puts("Even integer");
    case 1:
        puts("Odd integer");
}

```

- c) Il seguente codice deve ricevere in ingresso un intero e un carattere e stamparli. Supponete che l'utente scriva in input 100 A.

```

scanf("%d", &intValue);
charVal = getchar();
printf("Integer: %d\nCharacter: %c\n", intValue, charVal);

```

d) `for (x = .000001; x == .0001; x += .000001) {`
 `printf("%.7f\n", x);`
`}`

- e) Il seguente codice deve stampare gli interi dispari da 999 a 1.

```

for (x = 999; x >= 1; x += 2) {
    printf("%d\n", x);
}

```

- f) Il seguente codice deve stampare gli interi pari da 2 a 100.

```

counter = 2;
Do {
    if (counter % 2 == 0) {
        printf("%d\n", counter);
    }
    counter += 2;
} While (counter < 100);

```

- g) Il seguente codice deve sommare gli interi da 100 a 150 (supponete che `total` sia inizializzato a 0).

```

for (x = 100; x <= 150; ++x) {
    total += x;
}

```

4.6 Stabilite quali valori della variabile di controllo *x* sono stampati da ognuna delle seguenti istruzioni *for*.

- a) `for (int x = 2; x <= 13; x += 2) {
 printf("%d\n", x);
}`
- b) `for (int x = 5; x <= 22; x += 7) {
 printf("%d\n", x);
}`
- c) `for (int x = 3; x <= 15; x += 3) {
 printf("%d\n", x);
}`
- d) `for (int x = 1; x <= 5; x += 7) {
 printf("%d\n", x);
}`
- e) `for (int x = 12; x >= 2; x -= 3) {
 printf("%d\n", x);
}`

4.7 Scrivete delle istruzioni *for* che stampino le seguenti sequenze di valori.

- a) 1, 2, 3, 4, 5, 6, 7
- b) 3, 8, 13, 18, 23
- c) 20, 14, 8, 2, -4, -10
- d) 19, 27, 35, 43, 51

4.8 Cosa fa il seguente programma?

```

1 #include <stdio.h>
2
3 int main(void) {
4     int x = 0;
5     int y = 0;
6
7     // richiesta per l'input dell'utente
8     printf("%s", "Enter two integers in the range 1-20: ");
9     scanf("%d%d", &x, &y); // legge i valori per x e y
10
11    for (int i = 1; i <= y; ++i) { // conta da 1 a y
12
13        for (int j = 1; j <= x; ++j) { // conta da 1 a x
14            printf("%s", "@");
15        }
16
17        puts(""); // inizia una nuova riga
18    }
19 }
```

4.9 (*Somma di una sequenza di interi*) Scrivete un programma che sommi una sequenza di interi. Supponete che il primo intero letto con *scanf* specifichi il numero di valori che devono ancora essere inseriti. Il vostro programma deve leggere solo un valore a ogni esecuzione di *scanf*. Una tipica sequenza di input potrebbe essere

5 100 200 300 400 500

dove il 5 indica che i cinque valori successivi si devono sommare.

4.10 (Calcolare la media di una sequenza di interi) Scrivete un programma che calcoli e stampi la media di diversi numeri interi. Supponete che l'ultimo valore letto con `scanf` sia la sentinella 9999. Una tipica sequenza di input potrebbe essere

10 8 11 7 9 9999

che indica che va calcolata la media di tutti i valori che precedono 9999.

4.11 (*Trovare il valore più piccolo*) Scrivete un programma che trovi il più piccolo di diversi numeri interi. Supponete che il primo valore letto specifichi il numero dei restanti valori.

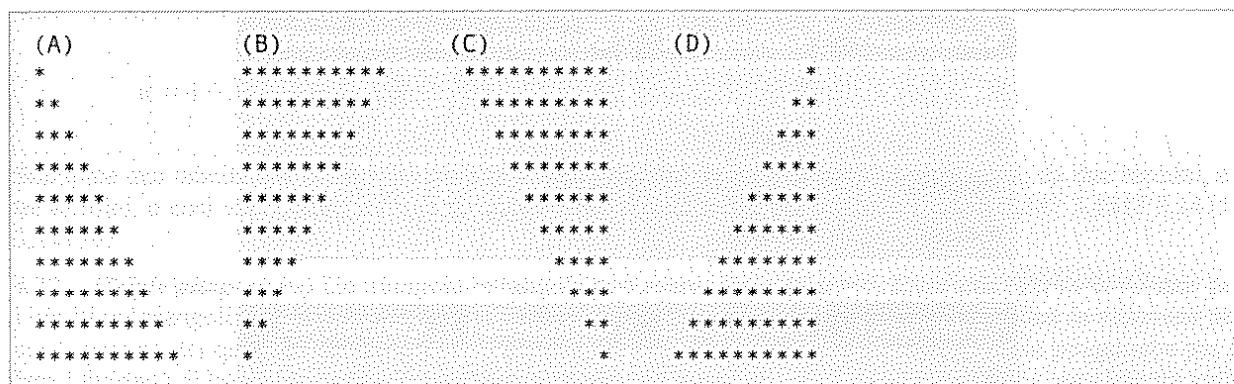
4.12 (*Calcolare la somma di numeri interi pari*) Scrivete un programma che calcoli e stampi la somma degli interi pari da 2 a 30.

4.13 (*Calcolare il prodotto di numeri interi dispari*) Scrivete un programma che calcoli e stampi il prodotto degli interi dispari da 1 a 15.

4.14 (Fattoriali) La funzione *fattoriale* è usata frequentemente nei problemi che riguardano la probabilità. Il fattoriale di un intero positivo n (scritto $n!$ e pronunciato “ n fattoriale”) è uguale al prodotto degli interi positivi da 1 a n . Scrivete un programma che calcoli i fattoriali degli interi da 1 a 5 e stampate i risultati in forma di tabella. Cosa potrebbe impedirvi di calcolare il fattoriale di 20?

4.15 (Programma modificato per l'interesse composto) Modificate il programma per l'interesse composto del Paragrafo 4.5 in modo da ripetere i suoi passi per i tassi di interesse del 5%, 6%, 7%, 8%, 9% e 10%. Usate un ciclo `for` per variare il tasso di interesse.

4.16 (Programma che stampa triangoli) Scrivete un programma che stampi separatamente i seguenti schemi, uno sotto l'altro. Usate dei cicli for per generare gli schemi. Tutti gli asterischi (*) devono essere stampati da una singola istruzione printf della forma printf("%s", "*"); (ciò fa sì che gli asterischi vengano stampati uno accanto all'altro). [Suggerimento: gli ultimi due schemi richiedono che ogni riga cominci con un numero appropriato di spazi.]



4.17 (Calcolare i limiti di credito) Riscuotere denaro diventa sempre più difficile in periodi di recessione, così le aziende possono diminuire i loro limiti di credito per evitare che i loro conti creditori (il denaro loro dovuto) diventino troppo grandi. In risposta a una prolungata recessione, un'azienda ha dimezzato i limiti di credito dei suoi clienti. In questo modo, se un particolare cliente aveva un limite di credito di \$2000, questo adesso è di \$1000. Se un cliente aveva un limite di credito di \$5000, ora è di \$2500. Scrivete un programma che analizzi lo status di tre clienti di questa azienda. Per ogni cliente vi sono dati:

- a) il numero di conto del cliente;
 - b) il limite di credito del cliente prima della recessione;
 - c) il saldo attuale di credito del cliente (cioè l'ammontare che il cliente deve all'azienda).

Il vostro programma deve calcolare e stampare il nuovo limite di credito per ciascun cliente e determinare (e stampare) quali clienti hanno il saldo attuale di credito che supera i loro nuovi limiti di credito.

4.18 (Programma che stampa un grafico a barre) Un'applicazione interessante dei computer è quella che disegna grafici e grafici a barre. Scrivete un programma che legga cinque numeri (ognuno tra 1 e 30). Per ogni numero letto, il vostro programma deve stampare una riga contenente quel numero di asterischi contigui. Per esempio, se il vostro programma legge il numero 7, deve stampare *****.

4.19 (Calcolo delle vendite) Un rivenditore online vende cinque differenti prodotti i cui prezzi al dettaglio sono mostrati nella tabella seguente.

Numero del prodotto	Prezzo al dettaglio
1	\$ 2.98
2	\$ 4.50
3	\$ 9.98
4	\$ 4.49
5	\$ 6.87

Scrivete un programma che legga una serie di coppie di numeri come segue:

- a) numero del prodotto;
- b) quantità venduta in un giorno.

Il vostro programma deve usare un'istruzione `switch` per permettervi di determinare il prezzo al dettaglio per ogni prodotto. Inoltre deve calcolare e stampare il valore al dettaglio totale di tutti i prodotti venduti nell'ultima settimana.

4.20 (Tabella di verità) Completate le seguenti tabelle di verità riempiendo ogni spazio vuoto con 0 o 1.

Condizione1	Condizione2	Condizione1 && Condizione2
0	0	0
0	nonzero	0
nonzero	0	----
nonzero	nonzero	----

Condizione1	Condizione2	Condizione1 Condizione2
0	0	0
0	nonzero	1
nonzero	0	----
nonzero	nonzero	----

Condizione1	!Condizione1
0	1
nonzero	----

4.21 Riscrivete il programma della Figura 4.2 per definire e inizializzare la variabile `counter` prima dell'istruzione `for`, quindi stampate il valore di `counter` dopo il termine del ciclo.

4.22 (*Calcolo della media dei voti*) Modificate il programma della Figura 4.5 in modo che calcoli la media dei voti per una classe.

4.23 (*Calcolo con interi dell'interesse composto*) Modificate il programma della Figura 4.4 in modo che usi soltanto numeri interi per calcolare l'interesse composto. [Suggerimento: trattate tutte le quantità monetarie come numeri interi di centesimi di dollaro. Poi "spezzate" il risultato nella sua porzione in dollari e nella sua porzione in centesimi usando, rispettivamente, gli operatori di divisione e di resto. Inserite poi un punto.]

4.24 Supponete che $i = 1$, $j = 2$, $k = 3$ e $m = 2$. Che cosa stampa ciascuna delle seguenti istruzioni?

- a) `printf("%d", i == 1);`
- b) `printf("%d", j == 3);`
- c) `printf("%d", i >= 1 && j < 4);`
- d) `printf("%d", m <= 99 && k < m);`
- e) `printf("%d", j >= i || k == m);`
- f) `printf("%d", k + m < j || 3 - j >= k);`
- g) `printf("%d", !m);`
- h) `printf("%d", !(j - m));`
- i) `printf("%d", !(k > m));`
- j) `printf("%d", !(j > k));`

4.25 (*Tabella di equivalenze fra decimali, binari, ottali ed esadecimali*) Scrivete un programma che stampi una tabella dei valori equivalenti binari, ottali ed esadecimali dei numeri decimali nell'intervallo da 1 a 256. Se non avete familiarità con questi sistemi di numerazione, leggete l'Appendice E (disponibile sulla piattaforma MyLab) prima di cimentarvi con questo esercizio. [Nota: potete stampare un intero come un valore ottale o esadecimale rispettivamente con le specifiche di conversione %o e %X, rispettivamente.]

4.26 (*Calcolo del valore di π*) Calcolate il valore di π in base alla serie infinita

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Stampate una tabella che mostri il valore di π approssimato da un termine di questa serie, da due termini, da tre termini, e così via. Quanti termini di questa serie dovete usare per ottenere i valori 3,14? 3,141? 3,1415? 3,14159?

4.27 (*Triple pitagoriche*) Un triangolo rettangolo può avere i lati tutti interi. L'insieme di tre valori interi per i lati di un triangolo rettangolo è chiamato tripla pitagorica. Questi tre lati devono soddisfare la relazione per cui la somma dei quadrati dei lati è uguale al quadrato dell'ipotenusa. Trovate tutte le triple pitagoriche per il lato 1 (`side1`), il lato 2 (`side2`) e l'ipotenusa, tutti non più grandi di 500. Usate un ciclo `for` annidato tre volte che tenti tutte le possibilità. Questo è un esempio di calcolo a "forza bruta". Per molte persone non è esteticamente attraente, ma vi sono molte ragioni per cui questa tecnica è importante. Innanzitutto, con la potenza di calcolo che aumenta a un tale ritmo fenomenale, soluzioni che avrebbero richiesto anni o perfino secoli per essere prodotte con la tecnologia di appena pochi anni fa si possono adesso calcolare in ore, minuti, secondi o anche in meno tempo. In secondo luogo, c'è un grande numero di problemi interessanti per i quali non vi è altro approccio algoritmico noto oltre a quello della pura e semplice forza bruta. In questo libro esamineremo molti tipi di metodologie per la risoluzione di problemi. Considereremo approcci a forza bruta per svariati e interessanti problemi.

4.28 (*Calcolo della paga settimanale*) Un'azienda paga i suoi impiegati come manager (che ricevono una retribuzione fissa settimanale), come lavoratori a ore (che ricevono una paga fissa all'ora per le prime 40 ore e "una volta e mezza" – cioè 1,5 volte la loro paga all'ora – per lo straordinario), lavoratori con provvigione (che ricevono \$250 più il 5,7% delle loro vendite settimanali lorde) o lavoratori a cottimo (che ricevono una quantità fissa di denaro per ogni articolo che producono; ogni lavoratore a cottimo di questa azienda lavora

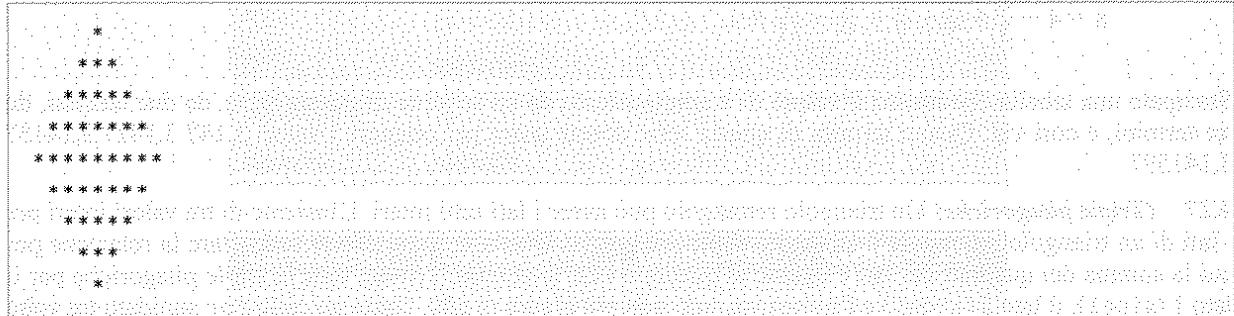
esclusivamente su un solo tipo di articolo). Scrivete un programma per calcolare la paga settimanale di ogni impiegato. Non conoscete in anticipo il numero degli impiegati. Ogni tipo di impiegato ha il proprio codice paga: i manager hanno il codice paga 1, i lavoratori a ore il 2, i lavoratori a provvigione il 3 e i lavoratori a cottimo il 4. Usate uno `switch` per calcolare la paga di ogni impiegato in base al codice paga. All'interno dello `switch`, richiedete all'utente di inserire i dati appropriati necessari al vostro programma per calcolare la paga di ogni impiegato in base al suo codice paga. [Nota: potete inserire valori di tipo `double` usando la specifica di conversione `%lf` con `scanf`.]

4.29 (Leggi di De Morgan) Abbiamo esaminato gli operatori logici `&&`, `||` e `!`. Le leggi di De Morgan aiutano a esprimere le espressioni logiche in modo più conveniente. Queste leggi stabiliscono che l'espressione `!(condizione1 && condizione2)` è logicamente equivalente all'espressione `(!condizione1 || !condizione2)`. Inoltre, l'espressione `!(condizione1 || condizione2)` è logicamente equivalente all'espressione `(!condizione1 && !condizione2)`. Usate le Leggi di De Morgan per scrivere espressioni equivalenti per ognuna delle seguenti espressioni, e poi scrivete un programma per mostrare che sia l'espressione originaria che l'espressione nuova sono equivalenti in ciascun caso.

- a) `!(x < 5) && !(y >= 7)`
- b) `!(a == b) || !(g != 5)`
- c) `!((x <= 8) && (y > 4))`
- d) `!((i > 4) || (j <= 6))`

4.30 (Sostituzione di switch con if...else) Riscrivete il programma della Figura 4.5 sostituendo l'istruzione `switch` con un'istruzione `if...else` annidata. Fate attenzione a trattare adeguatamente il caso `default`. Quindi riscrivete questa nuova versione sostituendo l'istruzione `if...else` annidata con una serie di istruzioni `if`. Anche qui fate attenzione a trattare adeguatamente il caso `default`. Questo esercizio dimostra che lo `switch` è una comodità e che qualsiasi istruzione `switch` può essere scritta soltanto con istruzioni di selezione singola.

4.31 (Programma che stampa un rombo) Scrivete un programma che stampi la seguente forma a rombo. Potete usare istruzioni `printf` che stampano un singolo asterisco (*) o un singolo spazio. Usate istruzioni `for` annidate e riducete al minimo il numero di istruzioni `printf`.



4.32 (Programma che stampa un rombo modificato) Modificate il programma scritto nell'Esercizio 4.31 per leggere un numero dispari compreso nell'intervallo da 1 a 19 per specificare il numero di righe nel rombo. Il vostro programma dovrebbe poi visualizzare un rombo della dimensione appropriata.

4.33 (Numerali romani equivalenti di valori decimali) Scrivete un programma che stampi una tabella dei numerali romani equivalenti dei numeri decimali nell'intervallo da 1 a 100.

4.34 Descrivete il processo che usereste per sostituire un ciclo `do...while` con un equivalente ciclo `while`. Quale problema si verifica quando cercate di sostituire un ciclo `while` con un equivalente ciclo `do...while?` Supponete che vi sia stato detto di rimuovere un ciclo `while` e di sostituirlo con un `do...while`. Quale ulteriore istruzione di controllo vi servirebbe e come la usereste per assicurarvi che il programma risultante si comporti esattamente come quello originario?

4.35 Un'obiezione alle istruzioni `break` e `continue` riguarda il fatto che ciascuna di esse non è strutturata. In realtà, le istruzioni `break` e `continue` possono sempre essere sostituite da istruzioni strutturate, sebbene fare ciò possa creare problemi. Descrivete in generale come rimuovereste da un ciclo di un programma una qualunque istruzione `break` e come la sostituireste con una equivalente strutturata. [Suggerimento: l'istruzione `break` termina un ciclo dal corpo del ciclo. L'altro modo per uscire è quello di far fallire il test di continuazione del ciclo. Nel test di continuazione del ciclo, considerate l'uso di un secondo test che indica "uscire subito a causa di una condizione `break`".] Usate la tecnica che avete sviluppato qui per rimuovere l'istruzione `break` dal programma della Figura 4.7.

4.36 Che cosa fa il seguente segmento di programma?

```

1  for (int i = 1; i <= 5; ++i) {
2      for (int j = 1; j <= 3; ++j) {
3          for (int k = 1; k <= 4; ++k) {
4              printf("%s", "*");
5          }
6          puts("");
7      }
8      puts("");
9  }
```

4.37 Descrivete in generale come rimuovereste una qualsiasi istruzione `continue` da un ciclo in un programma e come sostituireste quell'istruzione con una equivalente strutturata. Usate la tecnica che avete sviluppato qui per rimuovere l'istruzione `continue` dal programma della Figura 4.8.

4.38 (*La canzone “The Twelve Days of Christmas”*) Scrivete un programma che utilizzi le istruzioni di iterazione e l'istruzione `switch` per stampare la canzone “The Twelve Days of Christmas”. Va usata una sola istruzione `switch` per stampare il giorno (cioè, “first”, “second”, ecc.). Dovete poi usare un'istruzione `switch` separata per stampare il resto di ogni verso.

4.39 (*Limitazioni di numeri in virgola mobile per quantità monetarie*) Il Paragrafo 4.5 ha messo in guardia sull'uso di valori in virgola mobile per calcoli monetari. Provate questo esperimento: create una variabile `float` con il valore `1000000.00`. Poi aggiungete a tale variabile il valore letterale `float 0.12f`. Stampate il risultato usando `printf` e la specifica di conversione `%.2f`. Che cosa ottenete?

4.40 (*Crescita della popolazione mondiale*) La popolazione mondiale è considerevolmente aumentata nel corso dei secoli. La crescita continua potrebbe alla fine sfidare i limiti di aria respirabile, di acqua potabile, di terreni coltivabili e di altre risorse limitate. È evidente che negli ultimi anni la crescita è rallentata e che la popolazione mondiale potrebbe raggiungere il picco massimo nel corso di questo secolo per poi iniziare a calare. Per questo esercizio ricercate online articoli sull'aumento della popolazione mondiale. *Assicuratevi di prendere in esame vari punti di vista.* Ottenete stime per l'attuale popolazione mondiale e il suo tasso di crescita (la percentuale con cui è probabile che aumenti quest'anno). Scrivete un programma che calcoli la crescita della popolazione mondiale ogni anno per i prossimi 75 anni, *usando l'assunto semplificativo che il tasso della crescita attuale resterà costante.* Stampate i risultati in una tabella. La prima colonna dovrebbe visualizzare l'anno (dall'anno 1 all'anno 75), la seconda la popolazione mondiale prevista per la fine di quell'anno, e la terza l'aumento numerico nella popolazione mondiale che si verificherebbe quell'anno. Usando i vostri risultati, determinate l'anno in cui la popolazione sarebbe il doppio di oggi, qualora continuasse il tasso di crescita di quest'anno.

CAPITOLO

5

Sommario del capitolo

- 5.1 Introduzione
- 5.2 Modularizzazione dei programmi in C
- 5.3 Funzioni della libreria math
- 5.4 Funzioni
- 5.5 Definizioni di funzioni
- 5.6 Prototipi di funzioni: uno sguardo più approfondito
- 5.7 Pila delle chiamate delle funzioni e record di attivazione
- 5.8 File di intestazione
- 5.9 Passare gli argomenti per valore e per riferimento
- 5.10 Generazione di numeri casuali
- 5.11 Caso pratico sulla simulazione di numeri casuali: creare un gioco da casinò
- 5.12 Classi di memoria
- 5.13 Regole per il campo d'azione
- 5.14 Ricorsione
- 5.15 Esempio di utilizzo della ricorsione: la serie di Fibonacci
- 5.16 La ricorsione rispetto all'iterazione
- 5.17 Programmazione sicura in C – Generazione sicura di numeri casuali
- 5.18 Riepilogo

Funzioni

Obiettivi

- Costruire programmi in maniera modulare partendo da piccoli elementi chiamati funzioni.
- Usare comuni funzioni matematiche della Libreria Standard del C.
- Creare nuove funzioni.
- Comprendere in che modo i prototipi di funzioni aiutano il compilatore a garantire l'utilizzo corretto delle funzioni.
- Usare i meccanismi di passaggio delle informazioni tra le funzioni.
- Esaminare alcuni file di intestazione della Libreria Standard del C comunemente utilizzati.
- Imparare come il meccanismo di chiamata/ritorno delle funzioni è supportato dalla pila delle chiamate delle funzioni e dai record di attivazione.
- Creare un gioco da casinò usando tecniche di simulazione e generazione di numeri casuali.
- Comprendere in che modo una classe di memoria di un identificatore ne determini la permanenza in memoria, il campo d'azione e il collegamento.
- Scrivere e usare funzioni ricorsive, cioè funzioni che chiamano se stesse.
- Continuare la nostra presentazione della programmazione sicura in C con uno sguardo alla generazione sicura di numeri casuali.

5.1 Introduzione

La maggior parte dei programmi per computer che risolvono problemi reali ha dimensioni molto più grandi rispetto ai programmi presentati nei primi capitoli. L'esperienza ha dimostrato che il modo migliore per sviluppare e mantenere un programma di grandi dimensioni è quello di costruirlo partendo da elementi più piccoli, ognuno dei quali è più maneggevole del programma originario. Questa tecnica è chiamata **dividi e conquista**. Questo capitolo descrive alcune caratteristiche chiave del linguaggio C per la progettazione, l'implementazione, l'utilizzo e la manutenzione di programmi di grandi dimensioni.

5.2 Modularizzazione dei programmi in C

In C si utilizzano le **funzioni** per la modularizzazione dei programmi combinando la scrittura di nuove funzioni con l'utilizzo di funzioni preconfezionate della **Libreria Standard del C**. La Libreria Standard del C fornisce una ricca collezione di funzioni per eseguire comuni calcoli matematici, manipolazioni di stringhe, manipolazioni di caratteri, input/output e molte altre operazioni utili. Le funzioni preconfezionate facilitano il vostro lavoro perché forniscono molte delle capacità che vi servono.

Il C standard include il linguaggio C e la sua Libreria Standard; i compilatori del C implementano entrambi.¹ Le funzioni printf, scanf e pow che abbiamo usato nei capitoli precedenti provengono dalla Libreria Standard.



Evitate di reinventare la ruota

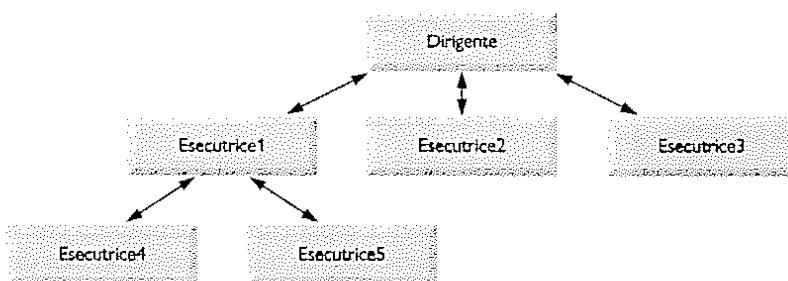
Familiarizzate con la ricca collezione di funzioni della Libreria Standard del C per ridurre i tempi di sviluppo del programma. Quando possibile, usate le funzioni standard invece di scrivere di nuove. Le funzioni della Libreria Standard del C sono scritte da esperti, ben collaudate ed efficienti. Inoltre, il loro utilizzo aiuta a rendere i programmi più portabili.

Definire le funzioni

È possibile definire funzioni per eseguire compiti specifici che possono essere utilizzate in più punti di un programma. Le istruzioni che definiscono la funzione vengono scritte una volta e sono nascoste alle altre funzioni. Come vedremo, tale occultamento è fondamentale per una buona ingegneria del software.

Chiamare funzioni e ritornare alle funzioni

Una funzione è **invocata** da una **chiamata di funzione**, che specifica il nome della funzione e fornisce le informazioni (come argomenti) che servono alla funzione per eseguire il suo compito designato.² Qualcosa di analogo a ciò è la forma gerarchica del management. Una dirigente (la **funzione che chiama o chiamante**) chiede a un'impiegata esecutrice (la **funzione chiamata**) di eseguire un compito e di fare rapporto al suo completamento. Per esempio, una funzione che deve stampare informazioni sullo schermo chiama la funzione esecutrice printf per eseguire quel compito, quindi printf stampa le informazioni e riferisce – o **ritorna** – alla funzione chiamante al completamento del compito. La funzione “dirigente” non sa come la funzione “esecutrice” esegue i suoi compiti. La funzione esecutrice può chiamare altre funzioni esecutrici e la dirigente può essere ignara di ciò. Il seguente diagramma mostra come una funzione dirigente comunichi con diverse funzioni esecutrici in maniera gerarchica:



Notate che Esecutrice1 agisce come funzione dirigente verso Esecutrice4 ed Esecutrice5. Le relazioni tra le funzioni possono essere anche di natura diversa rispetto alla struttura gerarchica illustrata in questa figura.

1. Alcune parti della Libreria Standard del C sono designate come opzionali e non sono disponibili in tutti i compilatori del C standard.
2. Nel Capitolo 7 analizzeremo i puntatori di funzioni. Vedrete che vi è anche la possibilità di chiamare una funzione tramite un puntatore di funzione e di passare funzioni ad altre funzioni.

✓ Autovalutazione

1. (*Completare*) I programmi sono solitamente scritti combinando la scrittura di nuove funzioni con l'utilizzo di funzioni preconfezionate disponibili nella _____.

Risposta: Libreria Standard del C.

2. (*Completare*) Le funzioni sono invocate da una _____ di funzione, che specifica il nome della funzione e fornisce le informazioni (come argomenti) che servono alla funzione per eseguire il suo compito designato.

Risposta: chiamata.

5.3 Funzioni della libreria math

Le funzioni della libreria math del C (file di intestazione `math.h`) permettono di eseguire comuni calcoli matematici, e in questo paragrafo ne useremo molte. Per calcolare e stampare la radice quadrata di `900.0` potete scrivere

```
printf("%.2f", sqrt(900.0));
```

Quando questa istruzione viene eseguita, chiama la funzione della libreria math `sqrt` per calcolare la radice quadrata di `900.0`, quindi stampa come risultato `30.00`. La funzione `sqrt` riceve un argomento di tipo `double` e restituisce un risultato di tipo `double`. In realtà, tutte le funzioni nella libreria math che restituiscono valori in virgola mobile restituiscono il tipo di dati `double`. Notate che i valori `double`, come i valori `float`, possono essere stampati usando la specifica di conversione `%f`. Potete memorizzare il risultato di una chiamata di una funzione in una variabile per un uso successivo come in:

```
double result = sqrt(900.0);
```

Gli argomenti di una funzione possono essere costanti, variabili o espressioni. Se `c = 13.0`, `d = 3.0` e `f = 4.0`, l'istruzione

```
printf("%.2f", sqrt(c + d * f));
```

calcola la radice quadrata di $13.0 + 3.0 * 4.0 = 25.0$ e stampa come risultato `5.00`.

La seguente tabella riassume alcune delle funzioni della libreria math del C. Nella tabella, le variabili `x` e `y` sono di tipo `double`. Lo standard C11 ha aggiunto funzionalità per numeri complessi tramite il file di intestazione `<complex.h>`.

Funzione	Descrizione	Esempio
<code>sqrt(x)</code>	radice quadrata di x	<code>sqrt(900.0)</code> è uguale a <code>30.0</code> <code>sqrt(9.0)</code> è uguale a <code>3.0</code>
<code>cbrt(x)</code>	radice cubica di x (solo per il C99 e il C11)	<code>cbrt(27.0)</code> è uguale a <code>3.0</code> <code>cbrt(-8.0)</code> è uguale a <code>-2.0</code>
<code>exp(x)</code>	funzione esponenziale e^x	<code>exp(1.0)</code> è uguale a <code>2.718282</code> <code>exp(2.0)</code> è uguale a <code>7.389056</code>
<code>log(x)</code>	logaritmo naturale di x (in base e)	<code>log(2.718282)</code> è uguale a <code>1.0</code> <code>log(7.389056)</code> è uguale a <code>2.0</code>
<code>log10(x)</code>	logaritmo di x (in base 10)	<code>log10(1.0)</code> è uguale a <code>0.0</code> <code>log10(10.0)</code> è uguale a <code>1.0</code> <code>log10(100.0)</code> è uguale a <code>2.0</code>
<code>fabs(x)</code>	valore assoluto di x come numero in virgola mobile	<code>fabs(13.5)</code> è uguale a <code>13.5</code> <code>fabs(0.0)</code> è uguale a <code>0.0</code> <code>fabs(-13.5)</code> è uguale a <code>13.5</code>



Funzione	Descrizione	Esempio
<code>ceil(x)</code>	arrotonda x all'intero più piccolo non minore di x	<code>ceil(9.2)</code> è uguale a 10.0 <code>ceil(-9.8)</code> è uguale a -9.0
<code>floor(x)</code>	arrotonda x all'intero più grande non maggiore di x	<code>floor(9.2)</code> è uguale a 9.0 <code>floor(-9.8)</code> è uguale a -10.0
<code>pow(x, y)</code>	x elevato alla potenza y (x^y)	<code>pow(2, 7)</code> è uguale a 128.0 <code>pow(9, .5)</code> è uguale a 3.0
<code>fmod(x, y)</code>	resto di x/y come numero in virgola mobile	<code>fmod(13.657, 2.333)</code> è uguale a 1.992
<code>sin(x)</code>	funzione trigonometrica seno di x (x in radianti)	<code>sin(0.0)</code> è uguale a 0.0
<code>cos(x)</code>	funzione trigonometrica coseno di x (x in radianti)	<code>cos(0.0)</code> è uguale a 1.0
<code>tan(x)</code>	funzione trigonometrica tangente di x (x in radianti)	<code>tan(0.0)</code> è uguale a 0.0

✓ Autovalutazione

- (Scelta multipla) Quale delle seguenti affermazioni a), b) o c) è falsa?
 - Chiamate una funzione scrivendo il suo nome seguito dal suo argomento, o da una lista di argomenti separati da virgole, tra parentesi.
 - La seguente istruzione calcola e memorizza la radice quadrata di 900.0:

```
double result = sqrt(900.0);
```

 - Per usare le funzioni della libreria math, dovete includere l'intestazione `math.h`.
 - Tutte le affermazioni precedenti sono vere.

Risposta: d.

- (Vero/Falso) Gli argomenti di una funzione possono essere costanti, variabili o espressioni. Se $c = 16.0$, $d = 4.0$ e $f = 5.0$, la seguente istruzione calcola e stampa la radice quadrata di 100.00:

```
printf("%.2f", sqrt(c + d * f));
```

Risposta: Falso. In realtà, calcola la radice quadrata di 36.0 e stampa come risultato 6.00.

5.4 Funzioni



Le funzioni permettono di rendere modulare un programma. In programmi contenenti molte funzioni, `main` è spesso implementata come un insieme di chiamate a funzioni che effettuano il grosso del lavoro del programma.

Funzionalizzare i programmi

Vi sono svariate motivazioni per "funzionalizzare" un programma. L'approccio **dividi e conquista** rende più fattibile lo sviluppo del programma. Un'altra motivazione è la costruzione di nuovi programmi usando funzioni esistenti. Tale **riusabilità del software** è un concetto chiave nei linguaggi di programmazione orientati agli oggetti derivati dal C, come C++, Java, C#, Objective-C e Swift.

Con una buona scelta dei nomi e delle definizioni delle funzioni si possono creare programmi partendo da funzioni standardizzate che eseguono compiti specifici, piuttosto che costruirli usando un codice sviluppato personalmente. Ciò è noto come **astrazione**. Usiamo l'astrazione ogni volta che usiamo funzioni della Libreria Standard come `printf`, `scanf` e `pow`. Una terza motivazione sta nell'evitare di ripetere codice in un programma. Impacchettare il codice come funzione ne permette l'esecuzione in altri punti del programma semplicemente chiamando la funzione.

Ogni funzione deve limitarsi a eseguire un singolo compito ben definito, e il nome della funzione deve esprimere quel compito. Questo facilita l'astrazione e favorisce la riusabilità del software. Se non riuscite a scegliere un nome conciso che indichi il compito della funzione, è possibile che la vostra funzione stia tentando di eseguire troppi compiti diversi. Di solito, è meglio spezzare una funzione di questo tipo in varie funzioni più piccole. Questa operazione è chiamata *scomposizione*.

✓ Autovalutazione

- (Completare)* Vi sono svariate motivazioni per “funzionalizzare” un programma. L’approccio dividi e conquista rende più fattibile lo sviluppo del programma. Un’altra motivazione è la _____ – usando funzioni esistenti come elementi costitutivi per creare nuovi programmi.

Risposta: riusabilità del software.

- (Vero/Falso)* Ogni funzione deve eseguire un ricco numero di compiti correlati, e il nome della funzione deve descriverli.

Risposta: *Falso*. In realtà, ogni funzione deve limitarsi a eseguire un singolo compito ben definito, e il nome della funzione deve esprimere quel compito. Questo facilita l’astrazione e favorisce la riusabilità del software.

5.5 Definizioni di funzioni

Ogni programma che abbiamo presentato consiste in una funzione chiamata `main` che chiama le funzioni della Libreria Standard per eseguire i suoi compiti. Vediamo adesso come scrivere funzioni personalizzate.

5.5.1 Funzione square

Considerate un programma che usa una funzione `square` per calcolare e stampare i quadrati dei numeri interi da 1 a 10 (Figura 5.1).

```

1 // fig05_01.c
2 // Creazione e uso di una funzione.
3 #include <stdio.h>
4
5 int square(int number); // prototipo di funzione
6
7 int main(void) {
8     // ripeti 10 volte e ogni volta calcola e stampa il quadrato di x
9     for (int x = 1; x <= 10; ++x) {
10         printf("%d ", square(x)); // chiamata della funzione
11     }
12
13     puts("");
14 }
15
16 // la definizione di square restituisce il quadrato del suo parametro
17 int square(int number) { // number e' una copia dell'argomento della funzione
18     return number * number; // restituisce il quadrato di number come valore int
19 }
```

1	4	9	16	25	36	49	64	81	100
---	---	---	----	----	----	----	----	----	-----

Figura 5.1 Creazione e uso di una funzione.

Chiamata della funzione square

La funzione `square` è **invocata** o **chiamata** nella funzione `main` all'interno dell'istruzione `printf` (riga 10):

```
printf("%d ", square(x)); // chiamata della funzione
```

La funzione `square` riceve una copia del valore dell'argomento `x` nel parametro `number` (riga 17). Poi `square` calcola `number * number` e restituisce il risultato alla riga 10 nella funzione `main` dove `square` è stata invocata. La riga 10 passa il risultato di `square` alla funzione `printf`, che stampa il risultato sullo schermo. Questo processo è ripetuto 10 volte, una volta per ogni iterazione dell'istruzione `for`.

Definizione della funzione square

La definizione della funzione `square` (righe 17–19) mostra che essa si aspetta un parametro intero `number`. La parola chiave `int` che precede il nome della funzione (riga 17) indica che `square` restituisce un risultato intero. L'**istruzione return** in `square` restituisce il risultato di `number * number` alla funzione chiamante.

 La scelta di nomi di funzioni e parametri significativi rende i programmi più leggibili e aiuta a evitare un uso eccessivo di commenti. I programmi dovrebbero essere scritti come raccolte di piccole funzioni. Ciò semplifica la scrittura, il debugging, la manutenzione, la modifica e il riutilizzo dei programmi.

 Una funzione che richiede un numero elevato di parametri potrebbe eseguire troppe attività, pertanto considerate di dividerla in funzioni più piccole che eseguano le attività separatamente. Il tipo di ritorno, il nome e la lista dei parametri della funzione dovrebbero essere contenuti su una singola riga, se possibile.

Variabili locali

Tutte le variabili definite nelle definizioni di funzione sono **variabili locali**: è possibile accedervi solo all'interno della funzione nella quale sono definite. La maggior parte delle funzioni ha **parametri** che permettono la comunicazione tra funzioni tramite argomenti nelle chiamate delle funzioni. I parametri di una funzione sono anche variabili locali di quella funzione.

Prototipo della funzione square

La riga 5

```
int square(int number); // prototipo di funzione
```

 è un **prototipo di funzione**. L'`int` tra parentesi informa il compilatore che `square` si aspetta di ricevere un valore intero dalla funzione chiamante. L'`int` alla sinistra del nome della funzione `square` informa il compilatore che `square` restituisce alla funzione chiamante un risultato intero. Dimenticare il punto e virgola alla fine di un prototipo di funzione è un errore di sintassi.

Il compilatore confronta la chiamata di `square` (riga 10) al suo prototipo per accertarsi che:

- il numero di argomenti sia corretto;
- i tipi degli argomenti siano corretti;
- i tipi degli argomenti siano in ordine corretto;
- il tipo di ritorno sia coerente con il contesto nel quale è chiamata la funzione.

Il prototipo della funzione, la prima riga della definizione della funzione e le chiamate della funzione dovrebbero concordare nel numero, nel tipo e nell'ordine degli argomenti e dei parametri. Il prototipo della funzione e l'intestazione della funzione devono avere lo stesso tipo di ritorno, che influisce su dove la funzione può essere chiamata. Per esempio, una funzione con il tipo di ritorno `void` non può essere utilizzata in un'istruzione di assegnazione per memorizzare un valore o in una chiamata a `printf` per stampare un valore. I prototipi di funzioni sono discussi in dettaglio nel Paragrafo 5.6.

Formato della definizione di una funzione

Il formato della definizione di una funzione è

```
tipo-del-valore-di-ritorno nome-della-funzione (lista-dei-parametri) {
    istruzioni
}
```

Il *nome-della-funzione* è qualsiasi identificatore valido. Il *tipo-del-valore-di-ritorno* è il tipo del risultato restituito alla funzione chiamante. Il *tipo-di-valore-di-ritorno* `void` indica che una funzione *non* restituisce alcun valore. Insieme, il *tipo-del-valore-di-ritorno*, il *nome-della-funzione* e la *lista-dei-parametri* costituiscono la cosiddetta **intestazione della funzione**.

 La *lista-dei-parametri* è un elenco separato da virgolette che specifica i parametri che la funzione riceve quando è chiamata. Se una funzione non riceve alcun valore, la *lista-dei-parametri* dovrebbe contenere la parola chiave `void`. Ogni parametro deve includere il suo tipo; altrimenti, si verifica un errore di compilazione.

 Porre un punto e virgola dopo la parentesi destra che chiude la *lista-dei-parametri* nella definizione di una funzione è un errore, in quanto si ridefinisce un parametro come variabile locale in una funzione. Sebbene non sia scorretto fare così, non usate gli stessi nomi per gli argomenti di una funzione e per i corrispondenti parametri nella definizione della funzione; ciò contribuisce a evitare ambiguità.

Corpo della funzione

 Le *istruzioni* all'interno delle parentesi graffe formano il **corpo della funzione**, che è anche detto blocco. Le variabili locali possono essere dichiarate in qualunque blocco e i blocchi possono essere annidati. Le funzioni non possono essere annidate: definire una funzione all'interno di un'altra funzione è un errore di sintassi.

Restituzione del controllo da una funzione

Vi sono tre modi per restituire il controllo da una funzione chiamata al punto in cui tale funzione è stata invocata. Se la funzione non restituisce alcun risultato, il controllo viene semplicemente restituito quando viene raggiunta la parentesi graffa che termina la funzione o quando si esegue l'istruzione

```
return;
```

Se la funzione restituisce un risultato, l'istruzione

```
return espressione;
```

restituisce alla funzione chiamante il valore di *espressione*.

Il tipo di ritorno della funzione `main`

Il valore di ritorno `int` della funzione `main` indica se il programma è stato eseguito correttamente. In versioni precedenti del C avremmo scritto esplicitamente

```
return 0; // 0 indica che il programma e' stato eseguito con successo
```

alla fine di `main`. Il C standard indica che `main` restituisce implicitamente 0 se omettete la precedente istruzione (come abbiamo fatto in questo libro). Potete far restituire esplicitamente a `main` valori diversi da zero per indicare che si è verificato un problema durante l'esecuzione del vostro programma. Per informazioni su come segnalare il fallimento di un programma, fate riferimento alla documentazione relativa al vostro particolare sistema operativo.

5.5.2 Funzione `maximum`

Ora consideriamo una funzione `maximum` personalizzata che restituisce il maggiore di tre numeri interi (Figura 5.2). Poi, essi vengono passati a `maximum` (riga 17), che determina l'intero maggiore. Questo valore è restituito a `main` con l'istruzione `return` in `maximum` (riga 32). L'istruzione `printf` alla riga 17 poi stampa il valore restituito da `maximum`.

```
1 // fig05_02.c
2 // Trovare il maggiore di tre interi.
3 #include <stdio.h>
4
5 int maximum(int x, int y, int z); // prototipo di funzione
6
7 int main(void) {
8     int number1 = 0; // primo intero inserito dall'utente
```

```

9     int number2 = 0; // secondo intero inserito dall'utente
10    int number3 = 0; // terzo intero inserito dall'utente
11
12    printf("%s", "Enter three integers: ");
13    scanf("%d%d%d", &number1, &number2, &number3);
14
15    // number1, number2 e number3 sono argomenti
16    // nella chiamata della funzione maximum
17    printf("Maximum is: %d\n", maximum(number1, number2, number3));
18 }
19
20 // Definizione della funzione maximum
21 int maximum(int x, int y, int z) {
22     int max = x; // supponi che x sia il maggiore
23
24     if (y > max) { // se y e' piu' grande di max,
25         max = y; // assegna y a max
26     }
27
28     if (z > max) { // se z e' piu' grande di max,
29         max = z; // assegna z a max
30     }
31
32     return max; // max e' il valore piu' grande
33 }
```

Enter three integers: 22 85 17
Maximum is: 85

Enter three integers: 47 32 14
Maximum is: 47

Enter three integers: 35 8 79
Maximum is: 79

Figura 5.2 Trovare il maggiore di tre interi.

La funzione inizialmente presuppone che il suo primo argomento (memorizzato nel parametro x) sia il maggiore e lo assegna a max (riga 22). Successivamente, l'istruzione if alle righe 24–26 determina se y è maggiore di max e, in caso affermativo, assegna y a max. Poi, l'istruzione if alle righe 28–30 determina se z è maggiore di max e, in caso affermativo, assegna z a max. Infine, la riga 32 restituisce max alla funzione chiamante.

✓ Autovalutazione

1. (*Scelta multipla*) La seguente riga di codice è _____.

- ```
int square(int y);
```
- Una definizione di funzione.
  - Un'istruzione di funzione.
  - Un prototipo di funzione.
  - Nessuno dei precedenti.

Risposta: c.

**2. (Scelta multipla)** Considerate la funzione `maximum` della Figura 5.2. Quale delle seguenti affermazioni è falsa?

- a) Il codice determina il maggiore dei tre valori interi.
- b) L'istruzione `return max;` restituisce il risultato alla funzione chiamante.
- c) Il codice alla riga 21 – `int maximum(int x, int y, int z)` – è comunemente chiamato intestazione della funzione.
- d) Se `int max = x;` (riga 22) venisse accidentalmente sostituito da `int max = y;` la funzione restituirebbe comunque lo stesso risultato.

**Risposta:** d) è falsa. La funzione restituirebbe poi in modo errato il maggiore dei valori contenuti nei soli parametri `y` e `z`.

## 5.6 Prototipi di funzioni: uno sguardo più approfondito

Un'importante caratteristica del C è il prototipo di una funzione, che è stato preso in prestito dal C++. Il compilatore usa i prototipi di funzioni per convalidare le chiamate delle funzioni. Le versioni precedenti del C *non* eseguivano questo genere di controllo, così era possibile chiamare le funzioni in maniera impropria senza che il compilatore individuasse gli errori. Simili chiamate potevano produrre errori irreversibili al momento dell'esecuzione o errori non irreversibili che causavano problemi subdoli, difficili da scoprire. I prototipi di funzioni correggono questa mancanza.

Dovreste includere prototipi di funzioni per tutte le funzioni, così da trarre vantaggio dalle capacità di controllo dei tipi del C. Usate la direttiva per il preprocessore `#include` per ottenere prototipi di funzioni da file di intestazione della Libreria Standard, file di intestazione di librerie di terze parti e intestazioni per funzioni sviluppate da voi o dai membri del vostro gruppo.

Il prototipo di funzione per `maximum` nella Figura 5.2 (riga 5) è

```
int maximum(int x, int y, int z); // prototipo di funzione.
```

Esso stabilisce che `maximum` riceve tre argomenti di tipo `int` e restituisce un risultato di tipo `int`. Notate che il prototipo di funzione (omettendo il punto e virgola) coincide con la prima riga della definizione di `maximum`. Includiamo i nomi dei parametri nei prototipi di funzioni per scopi di documentazione. Il compilatore ignora questi nomi, quindi anche il seguente prototipo è valido:

```
int maximum(int, int, int);
```

### Erri di compilazione

Una chiamata di funzione che non si accorda con il prototipo della funzione stessa genera un errore di compilazione. Viene generato un errore anche se non concordano tra loro il prototipo e la definizione della funzione. Per esempio, nella Figura 5.2, se il prototipo di funzione fosse stato scritto

```
void maximum(int x, int y, int z);
```

il compilatore avrebbe generato un errore, perché il tipo di ritorno `void` del prototipo di funzione sarebbe stato diverso dal tipo di ritorno `int` dell'intestazione della funzione.

### Coercione degli argomenti e “normali regole di conversione aritmetica”

Un'altra importante caratteristica dei prototipi di funzioni è la **coercione degli argomenti**, cioè la conversione implicita degli argomenti nel tipo appropriato. Per esempio, la funzione della libreria `math sqrt` può essere chiamata con un argomento intero anche se il prototipo della funzione in `<math.h>` specifica un parametro `double`; anche in questo caso la funzione opererà correttamente. La seguente istruzione valuta correttamente `sqrt(4)` e stampa `2.000`:

```
printf("%.3f\n", sqrt(4));
```

Il prototipo della funzione fa sì che il compilatore converte una copia del valore `int 4` nel valore `double 4.0` prima che venga passata a `sqrt`. In generale, i valori degli argomenti che non corrispondono precisamente ai tipi dei parametri nel prototipo della funzione vengono convertiti nel tipo adatto prima che la funzione sia

 chiamata. Queste conversioni possono portare a risultati scorretti se non si seguono le **normali regole di conversione aritmetica** del C. Queste regole specificano come i valori possono essere convertiti in altri tipi senza perdita di dati.

Nel nostro esempio relativo a `sqrt`, un `int` viene convertito automaticamente in un `double` senza che venga modificato il suo valore – `double` può rappresentare un intervallo di valori molto più grande di `int`. Tuttavia, la conversione di un `double` in un `int` fa sì che venga troncata la parte frazionaria di `double`, modificando così il valore originario. Convertire tipi interi grandi in tipi interi piccoli (per esempio, `long` in `short`) può anche modificare i valori.

### Espressioni con tipi misti

Le normali regole di conversione aritmetica sono gestite dal compilatore e si applicano a **espressioni con tipi misti**, cioè alle espressioni contenenti valori di più tipi di dati. In tali espressioni, il compilatore fa una copia temporanea dei valori da convertire, poi converte le *copie* nel tipo “più alto” nell’espressione: questo è noto come **promozione**. Per le espressioni con tipi misti contenenti almeno un valore in virgola mobile:

- se un valore è un `long double`, gli altri valori vengono convertiti in `long double`;
- se un valore è un `double`, gli altri valori vengono convertiti in `double`;
- se un valore è un `float`, gli altri valori vengono convertiti in `float`.

Se l’espressione con tipi misti contiene solo tipi interi, le normali conversioni aritmetiche specificano un insieme di regole di promozione intera.

Il Paragrafo 6.3.1 del documento del C standard specifica i dettagli completi degli operandi aritmetici e le normali regole di conversione aritmetica. Nella seguente tabella sono elencati i tipi di dati in virgola mobile e interi con le specifiche di conversione di ogni tipo per `printf` e `scanf`. Nella maggior parte dei casi, i tipi interi più in basso nella seguente tabella sono convertiti nei tipi più in alto.

| Tipo di dati                        | Specifiche di conversione per <code>printf</code> | Specifiche di conversione per <code>scanf</code> |
|-------------------------------------|---------------------------------------------------|--------------------------------------------------|
| <i>Tipi in virgola mobile</i>       |                                                   |                                                  |
| <code>long double</code>            | <code>%Lf</code>                                  | <code>%Lf</code>                                 |
| <code>double</code>                 | <code>%f</code>                                   | <code>%lf</code>                                 |
| <code>float</code>                  | <code>%f</code>                                   | <code>%f</code>                                  |
| <i>Tipi interi</i>                  |                                                   |                                                  |
| <code>unsigned long long int</code> | <code>%llu</code>                                 | <code>%llu</code>                                |
| <code>long long int</code>          | <code>%lld</code>                                 | <code>%lld</code>                                |
| <code>unsigned long int</code>      | <code>%lu</code>                                  | <code>%lu</code>                                 |
| <code>long int</code>               | <code>%ld</code>                                  | <code>%ld</code>                                 |
| <code>unsigned int</code>           | <code>%u</code>                                   | <code>%u</code>                                  |
| <code>int</code>                    | <code>%d</code>                                   | <code>%d</code>                                  |
| <code>unsigned short</code>         | <code>%hu</code>                                  | <code>%hu</code>                                 |
| <code>short</code>                  | <code>%hd</code>                                  | <code>%hd</code>                                 |
| <code>char</code>                   | <code>%c</code>                                   | <code>%c</code>                                  |

È possibile convertire un valore in un tipo più basso solamente assegnando esplicitamente il valore a una variabile di tipo più basso o usando un operatore cast. Gli argomenti sono convertiti nei tipi dei parametri specificati nel prototipo della funzione come se gli argomenti fossero assegnati direttamente a variabili di quei tipi.

Pertanto, se passiamo un `double` alla nostra funzione `square` della Figura 5.1, il `double` viene convertito in `int` (un tipo più in basso), e `square` solitamente restituisce un valore non corretto. Per esempio, `square(4.5)` restituisce 16, non 20.25.

 Convertire da un tipo di dati più alto nella gerarchia di promozione in un tipo più basso può cambiare il valore dei dati. In tali casi molti compilatori emettono avvisi.

#### Note sui prototipi di funzioni

 Se non vi è alcun prototipo per una funzione, il compilatore ne forma uno dalla prima occorrenza della funzione – o la definizione della funzione o una chiamata alla funzione. Ciò tipicamente fa emettere messaggi di avviso o messaggi di errore, a seconda del compilatore.

 Includevi sempre i prototipi per le funzioni definite o usate nel vostro programma, in modo da prevenire errori e avvisi in fase di compilazione.

 Un prototipo di funzione posto al di fuori della definizione di qualsiasi altra funzione si rivolge a tutte le chiamate alla funzione che compaiono *dopo* il prototipo della funzione. Un prototipo di funzione posto nel corpo di una funzione si rivolge solo alle chiamate in quella funzione fatte dopo di esso.

### ✓ Autovalutazione

1. (*Completare*) In un'espressione con tipi misti, il compilatore fa una copia temporanea dei valori da convertire, poi converte le copie nel tipo “più alto” nell'espressione: questo è noto come \_\_\_\_\_.

**Risposta:** promozione.

2. (*Scelta multipla*) Considerate il seguente prototipo di funzione per una funzione `maximum`:

```
int maximum(int x, int y, int z); // prototipo di funzione
```

Quale delle seguenti affermazioni è falsa?

- Afferma che `maximum` accetta tre argomenti di tipo `int` e restituisce un risultato di tipo `int`.
- È necessario includere i nomi dei parametri nei prototipi di funzioni.
- Un prototipo di funzione è spesso uguale all'intestazione della funzione, tranne per il fatto che l'intestazione non termina con un punto e virgola.
- Dimenticare il punto e virgola alla fine di un prototipo di funzione è un errore di sintassi.

**Risposta:** b) è falsa. I nomi dei parametri nei prototipi di funzioni servono solo per la documentazione. Il compilatore ignora questi nomi, pertanto `int maximum(int, int, int);` equivale al prototipo precedente.

## 5.7 Pila delle chiamate delle funzioni e record di attivazione

Per comprendere come il C esegua le chiamate delle funzioni, dobbiamo innanzitutto considerare una struttura di dati (cioè un insieme di dati correlati) nota come **pila** o **stack**. Pensate a una pila come a qualcosa di analogo a una pila di piatti. Quando si mette un piatto su una pila, normalmente lo si colloca in cima, operazione che chiameremo **push**. In modo simile, quando si toglie un piatto dalla pila, normalmente si prende quello in cima, operazione che chiameremo **pop**. Le pile sono note come **strutture di dati last-in, first-out (LIFO)**, ossia l'ultimo elemento inserito nella pila è il *primo* elemento a essere rimosso da essa.

### Pila delle chiamate delle funzioni

Un meccanismo che gli studenti di informatica devono comprendere bene è la **pila delle chiamate delle funzioni** (detta anche **pila di esecuzione del programma**). Questa struttura di dati, che lavora “dietro le quinte”, supporta il meccanismo di chiamata e ritorno delle funzioni. Come vedrete in questa sezione, la pila delle chiamate delle funzioni supporta anche la creazione, il mantenimento e la distruzione delle variabili locali di tutte le funzioni chiamate.

### Record di attivazione

Quando una funzione è chiamata, essa può chiamare altre funzioni, le quali possono a loro volta chiamarne altre; tutto avviene prima che una funzione torni alla funzione chiamante, restituendo a essa il controllo. Così, dobbiamo tenere traccia degli indirizzi di ritorno che servono a ogni funzione per tornare alla funzione che l'ha

chiamata. La pila delle chiamate delle funzioni è la struttura di dati perfetta per trattare queste informazioni. Ogni volta che una funzione chiama un'altra funzione, con un push un nuovo elemento viene inserito in cima alla pila. Questo elemento, chiamato **record di attivazione**, contiene l'*indirizzo di ritorno* che serve alla funzione chiamata per tornare alla funzione chiamante. Esso contiene anche alcune informazioni aggiuntive che presto esamineremo. Se la funzione chiamata torna alla funzione chiamante, con un pop il record di attivazione della chiamata della funzione viene estratto dalla cima della pila e il controllo si trasferisce all'indirizzo di ritorno memorizzato in quello stesso record di attivazione.

Ogni funzione chiamata trova sempre in *cima* alla pila delle chiamate le informazioni che le occorrono per tornare alla sua funzione chiamante. Se una funzione chiamata invoca un'altra funzione, con un push il record di attivazione per la chiamata della nuova funzione viene inserito in cima alla pila delle chiamate. In questo modo, l'indirizzo di ritorno richiesto dalla funzione chiamata più di recente per tornare alla sua funzione chiamante è ora posizionato in cima alla pila.

I record di attivazione hanno un'altra importante responsabilità. La maggior parte delle funzioni hanno variabili locali, che devono esistere mentre una funzione è in esecuzione e devono inoltre rimanere attive se la funzione chiama altre funzioni. Ma quando una funzione chiamata torna alla sua funzione chiamante, le variabili locali della funzione chiamata devono “sparire”. Il record di attivazione della funzione chiamata è un posto perfetto per memorizzare le variabili locali. Quel record di attivazione esiste solo finché la funzione chiamata è attiva. Quando quella funzione torna alla funzione chiamante – e non ha più bisogno delle sue variabili locali – con un pop il record di attivazione è rimosso dalla pila e quelle variabili locali non esistono più per il programma.

### Stack overflow

Naturalmente, la quantità di memoria in un computer è limitata, quindi se ne può utilizzare solo una certa quantità per memorizzare i record di attivazione nella pila delle chiamate delle funzioni. Se avvengono più chiamate di funzioni di quanti record di attivazione possono essere memorizzati nella pila delle chiamate, si verifica un errore *irreversibile* noto come **stack overflow**<sup>3</sup>.

### Pila delle chiamate delle funzioni in azione

Consideriamo ora come la pila delle chiamate supporta l'operazione di una funzione `square` chiamata da `main` (righe 8–12 della Figura 5.3).

```

1 // fig05_03.c
2 // Meccanismo della pila delle chiamate delle funzioni
3 // e dei record di attivazione con l'esempio di una funzione square.
4 #include <stdio.h>
5
6 int square(int x); // prototipo per la funzione square
7
8 int main() {
9 int a = 10; // valore in square (variabile locale in main)
10
11 printf("%d squared: %d\n", a, square(a)); // stampa un quadrato
12 }
13
14 // restituisce il quadrato di un intero
15 int square(int x) { // x e' una variabile locale
16 return x * x; // calcola il quadrato e restituisce il risultato
17 }
```

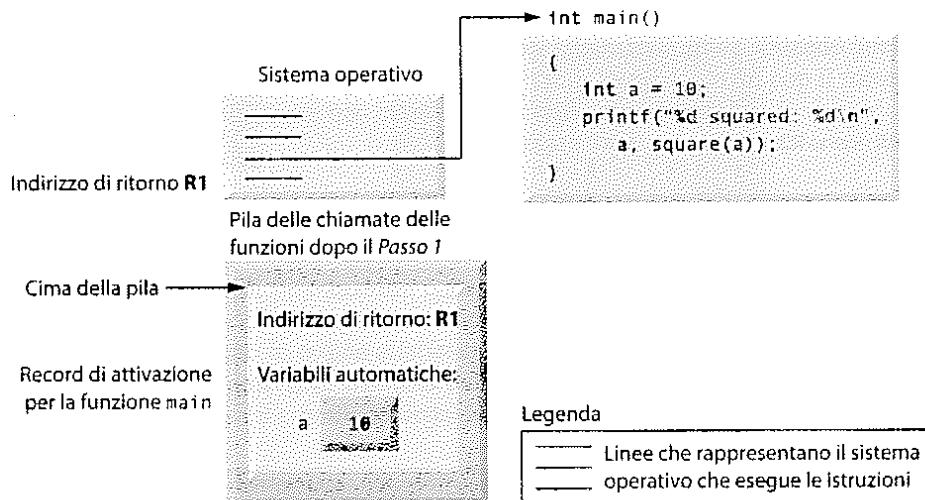
10 squared: 100

**Figura 5.3** Meccanismo della pila delle chiamate delle funzioni e dei record di attivazione con l'esempio di una funzione `square`.

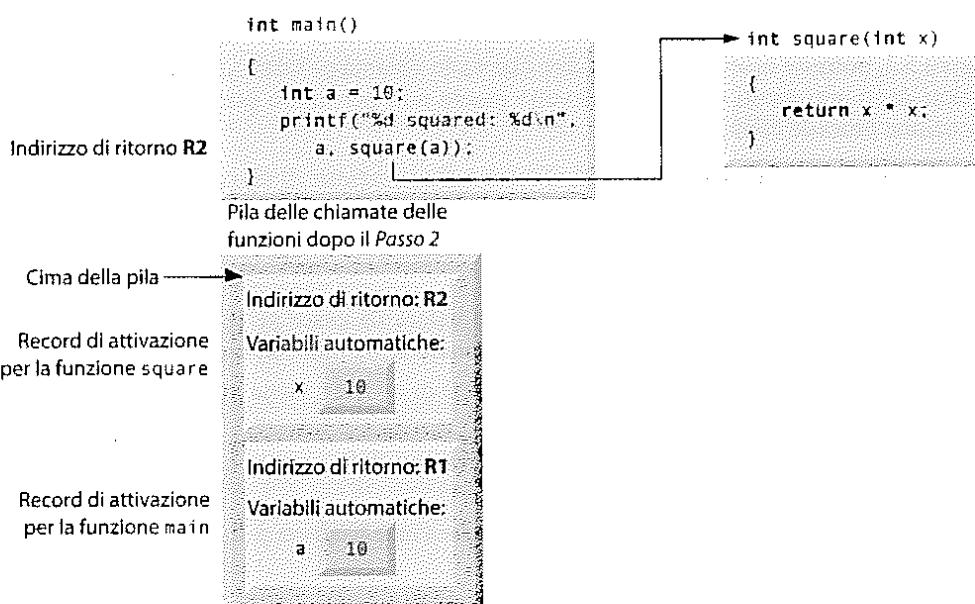
3. Da qui ha preso il nome [stackoverflow.com](http://stackoverflow.com), un popolare sito web per ottenere risposte a domande sulla programmazione.

**Passo 1: il sistema operativo invoca main per eseguire l'applicazione**

Per prima cosa il sistema operativo chiama `main`; ciò implica un push di un record di attivazione nella pila (come mostrato nel seguente diagramma). Il record di attivazione indica a `main` come tornare al sistema operativo (cioè come trasferire il controllo all'indirizzo di ritorno `R1`) e contiene lo spazio per la variabile locale `a` di `main`, che è inizializzata a 10.

**Passo 2: main invoca la funzione square per eseguire il calcolo**

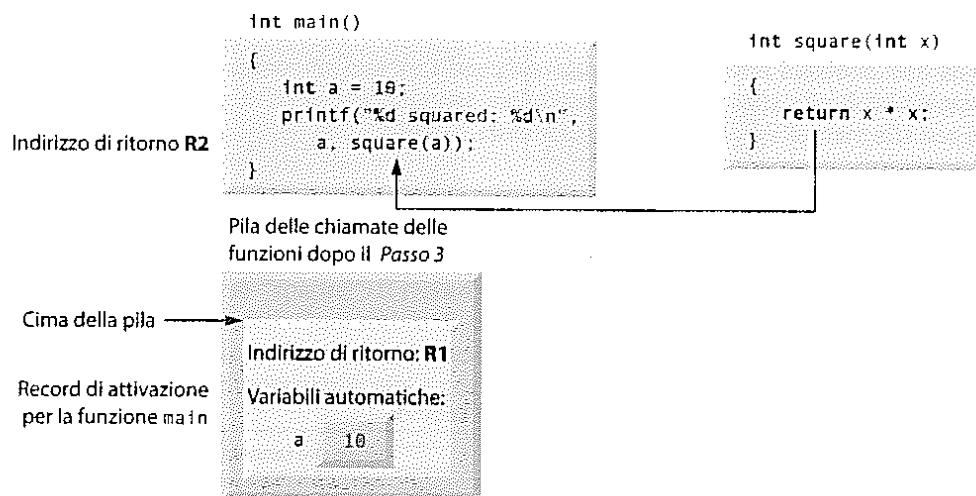
La funzione `main` – prima di tornare al sistema operativo – chiama ora la funzione `square` alla riga 11 della Figura 5.3. Questo causa un push di un record di attivazione per `square` (righe 15–17) nella pila delle chiamate delle funzioni, come mostrato nel seguente diagramma:



Questo record di attivazione contiene l'indirizzo di ritorno che serve a `square` per tornare a `main` (cioè `R2`) e la memoria per la variabile locale di `square` (cioè `x`).

**Passo 3: square restituisce il suo risultato a main**

Dopo che `square` calcola il quadrato del suo argomento, deve tornare a `main` – e non ha più bisogno della memoria per la sua variabile locale `x`. Così si effettua un `pop` dalla pila, fornendo a `square` l'indirizzo di ritorno in `main` (cioè `R2`) e perdendo la variabile locale di `square`. Il seguente diagramma mostra la pila delle chiamate delle funzioni *dopo* la rimozione del record di attivazione di `square`.



La funzione `main` stampa ora il risultato della chiamata a `square` (riga 11 della Figura 5.3). Quando si raggiunge la parentesi graffa destra di chiusura di `main`, il suo record di attivazione viene rimosso dalla pila, `main` riceve l'indirizzo che le occorre per tornare al sistema operativo (cioè, `R1` nel diagramma precedente). A questo punto, la memoria per la variabile locale di `main` (cioè `a`) non è più accessibile.

**Un difetto nella nostra discussione**

C'è un difetto nella discussione e nei diagrammi precedenti. Abbiamo mostrato `main` che invoca `square` e `square` che restituisce il suo risultato a `main`, ma, ovviamente, anche `printf` è una funzione. Studiando il codice della Figura 5.3, potreste essere propensi a dire che `main` chiama `printf`, quindi `printf` chiama `square`. Tuttavia, i valori degli argomenti di `printf` devono essere noti per intero prima di poter chiamare `printf`. Quindi l'esecuzione procede come segue.

1. Il sistema operativo chiama `main`, quindi il record di attivazione di `main` viene inserito con un `push` in cima alla pila.
2. `main` chiama `square`, quindi il record di attivazione di `square` viene inserito con un `push` in cima alla pila.
3. `square` esegue il calcolo e restituisce a `main` un valore da usare nella lista di argomenti di `printf`, quindi il record di attivazione di `square` viene rimosso dalla pila con un `pop`.
4. `main` chiama `printf`, quindi il record di attivazione di `printf` viene inserito con un `push` in cima alla pila.
5. `printf` stampa i suoi argomenti, poi restituisce il suo risultato a `main`, quindi il record di attivazione di `printf` viene rimosso dalla pila con un `pop`.
6. `main` termina, quindi il record di attivazione di `main` viene rimosso dalla pila con un `pop`.

**Strutture di dati**

Avete appena visto quanto sia preziosa la struttura a pila per implementare un meccanismo chiave in grado di supportare l'esecuzione dei programmi. Le strutture di dati hanno applicazioni molto importanti nell'ambito dell'informatica. Esamineremo pile, code, liste e alberi nel Capitolo 12.

## ✓ Autovalutazione

1. (*Completare*) Ogni volta che una funzione chiama un'altra funzione, con un push un nuovo elemento viene inserito in cima alla pila. Questo elemento, chiamato \_\_\_\_\_, contiene l'indirizzo di ritorno che serve alla funzione chiamata per tornare alla funzione chiamante.

**Risposta:** record di attivazione.

2. (*Vero/Falso*) Il record di attivazione della funzione chiamata è un posto perfetto per memorizzare le variabili locali. Quel record di attivazione esiste solo finché la funzione chiamata è attiva. Quando quella funzione torna alla funzione chiamante – e non ha più bisogno delle sue variabili locali – con un pop il record di attivazione è rimosso dalla pila e quelle variabili locali non esistono più per il programma.

**Risposta:** Vero.

## 5.8 File di intestazione

Ogni libreria standard ha un **file di intestazione** corrispondente, contenente i prototipi di funzioni per tutte le funzioni in quella libreria e le definizioni dei vari tipi di dati e delle costanti che servono a quelle funzioni. Nella seguente tabella sono elencati in ordine alfabetico alcuni file di intestazione della Libreria Standard che si possono includere nei programmi. Il C standard include ulteriori file di intestazione. Il termine “macro”, usato diverse volte nella tabella, sarà esaminato dettagliatamente nel Capitolo 14.

| File di intestazione                                          | Spiegazione                                                                                                                                                                           |
|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>File di intestazione usati o discussi in questo libro:</i> |                                                                                                                                                                                       |
| <assert.h>                                                    | Contiene informazioni per aggiungere istruzioni di diagnostica che agevolano il debugging (ricerca di errori e correzione) dei programmi.                                             |
| <ctype.h>                                                     | Contiene prototipi di funzioni che verificano certe proprietà dei caratteri e per le funzioni che si possono usare per convertire lettere minuscole in lettere maiuscole e viceversa. |
| <float.h>                                                     | Contiene i limiti per i valori in virgola mobile del sistema.                                                                                                                         |
| <limits.h>                                                    | Contiene i limiti per i valori interi del sistema.                                                                                                                                    |
| <math.h>                                                      | Contiene i prototipi di funzioni per le funzioni della libreria math.                                                                                                                 |
| <signal.h>                                                    | Contiene i prototipi di funzioni e le macro per gestire varie situazioni che possono insorgere durante l'esecuzione di un programma.                                                  |
| <stdarg.h>                                                    | Definisce le macro per trattare liste di argomenti per una funzione, il cui numero e i cui tipi non sono noti a priori.                                                               |
| <stdio.h>                                                     | Contiene i prototipi di funzioni per le funzioni della libreria standard di input/output, nonché le informazioni usate da queste.                                                     |
| <stdlib.h>                                                    | Contiene i prototipi di funzioni per convertire numeri in testo e testo in numeri, per allocare memoria, per trattare numeri casuali e per altre funzioni di utilità.                 |
| <string.h>                                                    | Contiene i prototipi di funzioni per le funzioni di elaborazione di stringhe.                                                                                                         |
| <time.h>                                                      | Contiene i prototipi di funzioni e i tipi per manipolare il tempo e le date.                                                                                                          |
| <i>Altri file di intestazione:</i>                            |                                                                                                                                                                                       |
| <errno.h>                                                     | Definisce le macro che sono utili per segnalare condizioni di errore.                                                                                                                 |



| File di intestazione | Spiegazione                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <locale.h>           | Contiene i prototipi di funzioni e altre informazioni che consentono di modificare il programma per l'ambiente locale nel quale viene eseguito. La nozione di ambiente locale consente a un sistema di elaborazione di trattare convenzioni diverse per esprimere dati come date, ore, quantità monetarie e grandi numeri dovunque nel mondo. |
| <setjmp.h>           | Contiene i prototipi di funzioni per le funzioni che permettono di non rispettare la normale sequenza di chiamata e ritorno di funzioni.                                                                                                                                                                                                      |
| <stddef.h>           | Contiene le definizioni di tipo comunemente usate dal C.                                                                                                                                                                                                                                                                                      |

Potete creare file di intestazione personalizzati e includerli usando la direttiva per il preprocessore `#include`. Per esempio, se il prototipo per la nostra funzione `square` fosse contenuto nel file di intestazione `square.h`, includeremmo questo file di intestazione nel nostro programma, usando la seguente direttiva in cima al programma stesso:

```
#include "square.h"
```

Il Paragrafo 14.2 fornisce ulteriori informazioni sull'inclusione di file di intestazione, come il motivo per cui le intestazioni definite dal programmatore sono racchiuse tra virgolette ("") anziché tra parentesi angolari (<>).

### ✓ Autovalutazione

1. (*Completare*) Il file di intestazione \_\_\_\_\_ contiene prototipi di funzioni per funzioni di elaborazione di stringhe.

Risposta: `<string.h>`.

2. (*Completare*) Il file di intestazione \_\_\_\_\_ contiene informazioni per aggiungere istruzioni di diagnostica che agevolano il debugging dei programmi.

Risposta: `<assert.h>`.

## 5.9 Passare gli argomenti per valore e per riferimento

In molti linguaggi di programmazione ci sono due modi per passare gli argomenti: il **passaggio per valore** e il **passaggio per riferimento**. Quando gli argomenti sono passati per valore, una copia del valore dell'argomento viene creata e passata alla funzione chiamata. Le modifiche alla copia non incidono sul valore della variabile originaria nella funzione chiamante. Quando un argomento è passato per riferimento, la funzione chiamante permette alla funzione chiamata di *modificare* il valore della variabile originaria.

Il passaggio per valore va usato ogni volta che la funzione chiamata non ha necessità di modificare il valore della variabile originaria della funzione chiamante. Questo evita **effetti secondari** (modificazioni delle variabili) che sono di grande impedimento allo sviluppo di sistemi software corretti e affidabili. Il passaggio per riferimento va usato solo con funzioni chiamate *fidate* che necessitano effettivamente di modificare la variabile originaria.

Nel C tutti gli argomenti sono passati per valore. Nel Capitolo 7 vedremo come ottenere il passaggio per riferimento. Nel Capitolo 6 vedremo che gli argomenti di tipo array sono automaticamente passati per riferimento per ragioni di prestazioni; nel Capitolo 7 vedremo che ciò non è una contraddizione. Concentriamoci per ora sul passaggio per valore.

### ✓ Autovalutazione

1. (*Vero/Falso*) Quando un argomento viene passato per valore, una copia del valore dell'argomento viene creata e passata alla funzione. Le modifiche alla copia vengono applicate anche al valore della variabile originaria nella funzione chiamante.

**Risposta:** *Falso*. Con il passaggio per valore, le modifiche alla copia non incidono sul valore della variabile originaria nella funzione chiamante.

2. (*Vero/Falso*) Il passaggio per riferimento va usato solo con funzioni chiamate *fidate* che necessitano effettivamente di modificare la variabile originaria.

**Risposta:** *Vero*.

## 5.10 Generazione di numeri casuali

Adesso ci concediamo un breve e, speriamo, piacevole diversivo con la *simulazione* e l'*esecuzione di un gioco*. In questo e nel prossimo paragrafo svilupperemo un programma per giochi ben strutturato che include molteplici funzioni personalizzate. Il programma usa funzioni e alcune delle istruzioni di controllo che abbiamo studiato. L'*elemento di casualità* può essere introdotto nelle applicazioni dei computer usando la funzione `rand`<sup>4</sup> del file di intestazione `<stdlib.h>`.

### Ottenere un valore intero casuale

Considerate la seguente istruzione:

```
int value = rand();
```

La funzione `rand` genera un numero intero tra 0 e `RAND_MAX` (una costante simbolica definita nel file di intestazione `<stdlib.h>`). Il C standard stabilisce che il valore di `RAND_MAX` deve essere almeno pari a 32.767, che è il valore massimo per un intero di due byte (cioè 16 bit). I programmi in questo paragrafo sono stati testati sul Visual C++ di Microsoft con un valore massimo `RAND_MAX` di 32.767 e su `gcc` di GNU e Clang di Xcode con un valore massimo `RAND_MAX` di 2.147.483.647. Se `rand` produce veramente interi a caso, ogni numero tra 0 e `RAND_MAX` ha un'uguale probabilità di essere scelto ogni volta che `rand` viene chiamata.

L'intervallo di valori prodotto direttamente da `rand` è spesso diverso da quello richiesto in una specifica applicazione. Per esempio, un programma che simula il lancio di una moneta può richiedere soltanto 0 per "testa" e 1 per "croce". Un programma per il lancio di dadi che simula un dado a sei facce richiede numeri interi casuali da 1 a 6.

### Lanciare un dado a sei facce

Per illustrare `rand`, sviluppiamo un programma (Figura 5.4) che simula 10 lanci di un dado a sei facce e stampa il valore ottenuto per ogni lancio.

```
1 // fig05_04.c
2 // Interi casuali con spostamento e variazione di scala prodotti da 1 + rand() % 6.
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7
8 for (int i = 1; i <= 10; ++i) {
9 printf("%d ", 1 + (rand() % 6)); // stampa valore del dado casuale
10 }
11
12 puts("");
13 }
```

6 6 5 5 6 5 1 1 5 3

6 6 5 5 6 5 1 1 5 3

**Figura 5.4** Interi casuali con spostamento e variazione di scala prodotti da  $1 + \text{rand}() \% 6$ .

4. La funzione `rand` della Libreria Standard del C è nota per essere "prevedibile", il che può creare opportunità di violazione della sicurezza. Ciascuna delle nostre piattaforme preferite offre un generatore di numeri casuali sicuro non standard. Ne parleremo nel Paragrafo 5.17.

Il prototipo della funzione `rand` si trova in `<stdlib.h>`. Alla riga 9, usiamo l'operatore di resto (%) congiuntamente a `rand` come segue

```
rand() % 6
```

per generare interi nell'intervallo da 0 a 5. Questa operazione è chiamata **variazione di scala**. Il numero 6 è noto come **fattore di scala**. Quindi effettuiamo uno **spostamento** dell'intervallo dei numeri generati aggiungendo 1 al nostro precedente risultato. L'output conferma che i risultati stanno nell'intervallo da 1 a 6 (l'ordine nel quale questi valori casuali sono scelti può variare in base al compilatore).

### Lancio di un dado a sei facce 60.000.000 di volte

Per verificare che questi numeri ricorrono approssimativamente con *uguale probabilità*, simuliamo 60.000.000 di lanci di un dado con il programma della Figura 5.5. Ogni intero da 1 a 6 deve comparire approssimativamente 10.000.000 di volte.

```
1 // fig05_05.c
2 // Lancio di un dado a sei facce 60.000.000 di volte.
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7 int frequency1 = 0; // contatore per il valore 1
8 int frequency2 = 0; // contatore per il valore 2
9 int frequency3 = 0; // contatore per il valore 3
10 int frequency4 = 0; // contatore per il valore 4
11 int frequency5 = 0; // contatore per il valore 5
12 int frequency6 = 0; // contatore per il valore 6
13
14 // ripeti 60.000.000 di volte e riepiloga i risultati
15 for (int roll = 1; roll <= 60000000; ++roll) {
16 int face = 1 + rand() % 6; // numero casuale da 1 a 6
17
18 // determina il valore di face e incrementa il contatore appropriato
19 switch (face) {
20 case 1: // valore 1
21 ++frequency1;
22 break;
23 case 2: // valore 2
24 ++frequency2;
25 break;
26 case 3: // valore 3
27 ++frequency3;
28 break;
29 case 4: // valore 4
30 ++frequency4;
31 break;
32 case 5: // valore 5
33 ++frequency5;
34 break;
35 case 6: // valore 6
36 ++frequency6;
37 break; // opzionale
38 }
39 }
```

```

40
41 // stampa i risultati in formato tabellare
42 printf("%s%13s\n", "Face", "Frequency");
43 printf(" 1%13d\n", frequency1);
44 printf(" 2%13d\n", frequency2);
45 printf(" 3%13d\n", frequency3);
46 printf(" 4%13d\n", frequency4);
47 printf(" 5%13d\n", frequency5);
48 printf(" 6%13d\n", frequency6);
49 }

```

| Face | Frequency |
|------|-----------|
| 1    | 9999294   |
| 2    | 10002929  |
| 3    | 9995360   |
| 4    | 10000409  |
| 5    | 10005206  |
| 6    | 9996802   |

Figura 5.5 Lancio di un dado a sei facce 60.000.000 di volte.

Come mostra l'output del programma, con una variazione di scala e uno spostamento dell'intervallo abbiamo usato la funzione `rand` per simulare realisticamente il lancio di un dado a sei facce. Notate l'uso della specifica di conversione `%s` per stampare le stringhe di caratteri "Face" e "Frequency" come intestazioni delle colonne (riga 42). Dopo aver studiato gli array nel Capitolo 6, mostreremo come sostituire elegantemente questa istruzione `switch` di 26 righe con un'istruzione di una singola riga.

### Randomizzazione del generatore di numeri casuali

Una nuova esecuzione del programma della Figura 5.4 produce

```
6 6 5 5 6 5 1 1 5 3
```

Questa è l'esatta sequenza di valori mostrata nella Figura 5.4. Come possono questi numeri essere casuali? Paradossalmente, questa *ripetibilità* è una caratteristica importante della funzione `rand`. Quando si effettua il debugging di un programma, questa ripetibilità è essenziale per provare che le correzioni di un programma operino correttamente.

La funzione `rand` genera in realtà **numeri pseudocasuali**. Chiamare ripetutamente `rand` produce una sequenza di numeri che appaiono casuali. Tuttavia, la sequenza ripete se stessa ogni volta che viene eseguito il programma. Una volta che si è effettuato il debugging completo del programma, è possibile far sì che esso produca una sequenza differente di numeri casuali per ogni esecuzione.

Tale operazione è chiamata **randomizzazione** ed è realizzata con la funzione `srand` della Libreria Standard. La funzione `srand` riceve un argomento `int` e fornisce un **seme** alla funzione `rand` per generare una sequenza diversa di numeri casuali per ogni esecuzione del programma.

Illustriamo la funzione `srand` nella Figura 5.6. Il prototipo di funzione per `srand` si trova in `<stdlib.h>`.

```

1 // fig05_06.c
2 // Randomizzare il programma del lancio del dado.
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7 printf("%s", "Enter seed: ");
8 int seed = 0; // numero usato come seme per il generatore di numeri casuali
9 scanf("%d", &seed);

```

```

10
11 srand(seed); // fornisci il seme al generatore di numeri casuali
12
13 for (int i = 1; i <= 10; ++i) {
14 printf("%d ", 1 + (rand() % 6)); // stampa il valore del dado casuale
15 }
16
17 puts("");
18 }
```

Enter seed: 67  
6 1 4 6 2 1 6 1 6 4

Enter seed: 867  
2 4 6 1 6 1 1 3 6 2

Enter seed: 67  
6 1 4 6 2 1 6 1 6 4

**Figura 5.6** Randomizzare il programma del lancio del dado.

Eseguiamo il programma diverse volte e osserviamo i risultati. Notate che ogni volta che il programma viene eseguito si ottiene una sequenza di numeri casuali differente, a condizione che sia fornito un seme differente. I primi e gli ultimi output utilizzano lo stesso valore del seme, quindi mostrano gli stessi risultati.

Per randomizzare senza inserire ogni volta un seme, usate l'istruzione

```
srand(time(NULL));
```

Questo fa sì che il computer legga il suo orologio per ottenere automaticamente il valore per il seme. La funzione `time` restituisce il numero dei secondi che sono trascorsi dalla mezzanotte dell'1 gennaio 1970. Questo valore è convertito in un intero e usato come seme per il generatore di numeri casuali. Il prototipo di funzione per `time` è in `<time.h>`. Diremo di più su `NULL` nel Capitolo 7.

### Generalizzazione della variazione di scala e dello spostamento dell'intervallo per i numeri casuali

I valori generati direttamente da `rand` sono sempre nell'intervallo:

$$0 \leq \text{rand}() \leq \text{RAND\_MAX}$$

Come sapete, l'istruzione seguente simula il lancio di un dado a sei facce:

```
int face = 1 + rand() % 6;
```

Questa istruzione assegna sempre un valore intero (a caso) alla variabile `face` nell'intervallo  $1 \leq \text{face} \leq 6$ . L'ampiezza di questo intervallo (cioè il numero di interi consecutivi nell'intervallo) è 6 e il numero iniziale dell'intervallo è 1. Riferendoci all'istruzione precedente, vediamo che l'ampiezza è determinata dal numero usato per scalare `rand` con l'operatore di resto (cioè 6), e il numero iniziale dell'intervallo è uguale al numero che è aggiunto a `rand % 6` (cioè 1). Possiamo generalizzare questo risultato come segue:

```
int n = a + rand() % b;
```

dove

- `a` è il **valore di spostamento** (che è uguale al primo numero nell'intervallo desiderato di interi consecutivi);

- $b$  è il *fattore di scala* (che è uguale all'ampiezza dell'intervallo desiderato di interi consecutivi).

Negli esercizi sceglierete a caso interi da insiemi di valori che non sono intervalli di interi consecutivi.

### Autovalutazione

- (Completere)* La \_\_\_\_\_ è una caratteristica importante della funzione `rand`. Quando si effettua il debugging di un programma, questa caratteristica è essenziale per provare che le correzioni di un programma operino correttamente.

**Risposta:** ripetibilità.

- (Vero/Falso)* Se `rand` produce veramente interi a caso, ogni numero tra 0 e `RAND_MAX` ha un'uguale probabilità di essere scelto ogni volta che `rand` viene chiamata.

**Risposta:** Vero.

- (Completere)* Una volta che si è effettuato il debugging completo del programma, è possibile far sì che esso produca una sequenza differente di numeri casuali per ogni esecuzione. Tale operazione è chiamata randomizzazione ed è realizzata con la funzione \_\_\_\_\_ della Libreria Standard.

**Risposta:** `srand`.

## 5.11 Caso pratico sulla simulazione di numeri casuali: creare un gioco da casinò

In questo paragrafo simuliamo il popolare gioco di dadi noto come "Craps". Le regole del gioco sono semplici:

*Un giocatore lancia due dadi. Ogni dado ha sei facce. Queste facce contengono 1, 2, 3, 4, 5 e 6 pallini. Quando i dadi si fermano, si calcola la somma dei pallini sulle due facce superiori. Se al primo lancio la somma è 7 o 11, il giocatore vince. Se al primo lancio la somma è 2, 3 o 12 (chiamata "craps"), il giocatore perde (cioè vince il banco). Se al primo lancio la somma è 4, 5, 6, 8, 9 o 10, quella somma diventa il "punteggio" del giocatore. Per vincere si devono continuare a lanciare i dadi finché si "fa il proprio punteggio". Il giocatore perde se ottiene come somma 7 prima di fare il proprio punteggio.*

La Figura 5.7 simula il gioco Craps e mostra diversi esempi di esecuzione.

```

1 // fig05_07.c
2 // Simulazione del gioco Craps.
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h> // contiene il prototipo per la funzione time
6
7 enum Status {CONTINUE, WON, LOST}; // le costanti rappresentano lo stato del gioco
8
9 int rollDice(void); // prototipo di funzione di rollDice
10
11 int main(void) {
12 srand(time(NULL)); // randomizza sulla base dell'ora corrente
13
14 int myPoint = 0; // il giocatore deve fare questo punteggio per vincere
15 enum Status gameStatus = CONTINUE; // puo' contenere CONTINUE, WON o LOST
16 int sum = rollDice(); // primo lancio dei dadi
17
18 // determina lo stato del gioco in base alla somma dei dadi
19 switch(sum) {
20 // si vince al primo lancio
21 case 7: // si vince con 7

```

```

22 case 11: // si vince con 11
23 gameStatus = WON;
24 break;
25 // si perde al primo lancio
26 case 2: // si perde con 2
27 case 3: // si perde con 3
28 case 12: // si perde con 12
29 gameStatus = LOST;
30 break;
31 // ricorda il punteggio
32 default:
33 gameStatus = CONTINUE; // il giocatore continua a lanciare
34 myPoint = sum; // ricorda il punteggio
35 printf("Point is %d\n", myPoint);
36 break; // opzionale
37 }
38
39 // finche' il gioco non si concludee
40 while (CONTINUE == gameStatus) { // il gioco continua
41 sum = rollDice(); // lancia di nuovo i dadi
42
43 // determina lo stato del gioco
44 if (sum == myPoint) { // si vince facendo il punteggio
45 gameStatus = WON;
46 }
47 else if (7 == sum) { // si perde ottenendo 7
48 gameStatus = LOST;
49 }
50 }
51
52 // stampa il messaggio di vittoria o di perdita
53 if (WON == gameStatus) { // il giocatore ha vinto?
54 puts("Player wins");
55 }
56 else { // il giocatore ha perso
57 puts("Player loses");
58 }
59 }
60
61 // lancia i dadi, calcola la somma e stampa i risultati
62 int rollDice(void) {
63 int die1 = 1 + (rand() % 6); // valore a caso per il primo dado
64 int die2 = 1 + (rand() % 6); // valore a caso per il secondo dado
65
66 // stampa i risultati di questo lancio
67 printf("Player rolled %d + %d = %d\n", die1, die2, die1 + die2);
68 return die1 + die2; // restituisci la somma dei dadi
69 }
```

*Il giocatore vince al primo lancio:*

```

Player rolled 5 + 6 = 11
Player wins
```

*Il giocatore vince a un lancio successivo:*

```
Player rolled 4 + 1 = 5
Point is 5
Player rolled 6 + 2 = 8
Player rolled 2 + 1 = 3
Player rolled 3 + 2 = 5
Player wins
```

*Il giocatore perde al primo lancio:*

```
Player rolled 1 + 1 = 2
Player loses
```

*Il giocatore perde a un lancio successivo:*

```
Player rolled 6 + 4 = 10
Point is 10
Player rolled 3 + 4 = 7
Player loses
```

**Figura 5.7** Simulazione del gioco Craps.

Nelle regole del gioco notate che il giocatore deve tirare due dadi al primo lancio e in tutti i lanci successivi. Definiamo una funzione `rollDice` per simulare il lancio dei dadi e calcolare e stampare la loro somma. La funzione `rollDice` è definita una sola volta, ma è chiamata in due punti diversi nel programma (righe 16 e 41). La funzione non riceve alcun argomento, così abbiamo inserito `void` nella lista dei parametri (riga 62) e nel prototipo di funzione (riga 9). La funzione `rollDice` restituisce la somma dei due dadi, così nella sua intestazione e nel suo prototipo di funzione è indicato un tipo di ritorno `int`.

### Enumerazioni

Il gioco è ragionevolmente articolato. Il giocatore può vincere o perdere al primo lancio o a un qualunque lancio successivo. La variabile `gameStatus` di un nuovo tipo – `enum Status` – memorizza lo stato corrente. La riga 7 crea un nuovo tipo chiamato **enumerazione**. Un'enumerazione, introdotta dalla parola chiave `enum`, è un insieme di costanti intere rappresentate da identificatori. Le **costanti di enumerazione** aiutano a rendere i programmi più leggibili e più facili da mantenere. I valori in un `enum` iniziano con 0 e sono incrementati di 1. Alla riga 7, la costante `CONTINUE` ha il valore 0, `WON` ha il valore 1 e `LOST` ha il valore 2. È anche possibile assegnare un valore intero a ciascun identificatore in un `enum` (vedi Capitolo 10). Gli identificatori in un'enumerazione devono essere unici, ma i valori possono essere duplicati. Usate solo lettere maiuscole nei nomi delle costanti di enumerazione per far sì che esse risaltino in un programma e per indicare che *non* sono delle variabili.

Quando si vince nel gioco, `gameStatus` assume il valore `WON`. Quando si perde, `gameStatus` assume il valore `LOST`. Diversamente, `gameStatus` assume il valore `CONTINUE`, e il gioco continua.

### Fine del gioco al primo lancio

Se il gioco finisce dopo il primo lancio, `gameStatus` non ha il valore `CONTINUE`, quindi il programma procede con l'istruzione `if...else` alle righe 53–58, che stampa "Player wins" se `gameStatus` è `WON`, altrimenti "Player loses".

### Fine del gioco a un lancio successivo

Dopo il primo lancio, se il gioco *non* è finito, il valore di sum viene salvato in myPoint. L'esecuzione procede con l'istruzione `while` perché gameStatus è CONTINUE. A ogni iterazione del `while`, rollDice è invocata per produrre un nuovo valore di sum:

- se sum coincide con myPoint, gameStatus assume il valore WON, il ciclo `while` termina, l'istruzione `if...else` stampa "Player wins" e l'esecuzione termina;
- se sum è uguale a 7 (riga 47), gameStatus assume il valore LOST, il ciclo `while` termina, l'istruzione `if...else` stampa "Player loses" e l'esecuzione termina.

### Architettura di controllo

Osservate l'architettura di controllo del programma. Abbiamo usato due funzioni – `main` e `rollDice` – e le istruzioni `switch`, `while` e `if...else` annidata.

### Esercizi correlati

Il caso pratico della creazione di un gioco da casinò è supportato dai seguenti esercizi:

- Esercizio 5.47 (Modifiche al gioco Craps).
- Esercizio 6.20 (Statistiche del gioco Craps).

### ✓ Autovalutazione

1. (*Completare*) Un'enumerazione, introdotta dalla parola chiave \_\_\_\_\_, è un insieme di costanti intere rappresentate da identificatori.

Risposta: enum.

2. (*Completare*) Nella seguente istruzione

```
enum Status {CONTINUE, WON, LOST};
```

i valori di CONTINUE, WON e LOST sono \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.

Risposta: 0, 1 e 2.

## 5.12 Classi di memoria

Nei Capitoli 2–4, abbiamo usato identificatori per i nomi delle variabili. Attributi delle variabili sono *nome*, *tipo*, *dimensione* e *valore*. In questo capitolo usiamo gli identificatori anche come nomi per le funzioni definite dall'utente. In realtà, ogni identificatore in un programma ha altri attributi, quali la classe di memoria, la permanenza in memoria, il campo d'azione e il collegamento.

Il C fornisce gli **specificatori della classe di memoria** `auto`, `register`,<sup>5</sup> `extern` e `static`.<sup>6</sup> Una **classe di memoria** determina la permanenza in memoria, il campo d'azione e il collegamento di un identificatore. La **permanenza in memoria** è il periodo durante il quale un identificatore esiste nella memoria. Alcuni esistono per breve tempo, alcuni sono ripetutamente creati e distrutti e altri esistono per l'intera esecuzione del programma. Il **campo d'azione** (in inglese "scope") determina dove l'identificatore può essere usato nel programma. Alcuni si possono menzionare per tutto il programma, altri solo da porzioni di esso. Per un programma con diversi file sorgente, il **collegamento** di un identificatore determina se l'identificatore è conosciuto soltanto nel file sorgente corrente o in qualunque file sorgente con le opportune dichiarazioni. Questo paragrafo esamina le classi di memoria e la permanenza in memoria. Il Paragrafo 5.13 esamina il campo d'azione. Il Capitolo 15 esamina il collegamento di un identificatore e la programmazione con più file sorgente.

5. La parola chiave `register` è arcaica e non dovrebbe essere usata.

6. Il C11 ha aggiunto lo specificatore della classe di memoria `_Thread_local`, che va oltre lo scopo di questo libro.

### Variabili locali e permanenza in memoria automatica

Gli specificatori della classe di memoria si suddividono tra **permanenza in memoria automatica** e **permanenza in memoria statica**. La parola chiave **auto** è usata per dichiarare le variabili con permanenza in memoria automatica. Tali variabili sono create quando il controllo del programma entra nel blocco nel quale sono definite; esse esistono finché il blocco è attivo e sono distrutte quando il controllo del programma esce dal blocco.

Solo le variabili possono avere permanenza in memoria automatica. Le variabili locali di una funzione (quelle dichiarate nella lista dei parametri o nel corpo della funzione) hanno permanenza in memoria automatica per default, quindi la parola chiave **auto** si usa raramente. La permanenza in memoria automatica è un mezzo per risparmiare memoria poiché le variabili locali esistono solo quando ce n'è necessità. Chiameremo le variabili con permanenza in memoria automatica semplicemente variabili locali.

### Classe di memoria statica

Le parole chiave **extern** e **static** dichiarano identificatori per variabili e funzioni con *permanenza in memoria statica*. Gli identificatori con permanenza in memoria statica esistono dal momento in cui il programma inizia l'esecuzione fino a che il programma termina. Per le variabili **static** la memoria è allocata e inizializzata *soltanto una volta, prima che il programma inizi l'esecuzione*. Per le funzioni, il nome della funzione esiste quando il programma inizia l'esecuzione. Tuttavia, anche se questi nomi esistono dall'avvio dell'esecuzione del programma, ciò non significa che siano sempre accessibili. La permanenza in memoria e il campo d'azione (*dove* si può usare un nome) sono aspetti separati, come vedremo nel Paragrafo 5.13.

Vi sono diversi tipi di identificatori con permanenza in memoria statica: gli *identificatori esterni* (come le variabili globali e i nomi di funzione) e le variabili locali dichiarate con lo specificatore della classe di memoria **static**. Le variabili globali e i nomi di funzione sono della classe di memoria **extern** per default. Le variabili globali sono create ponendo le dichiarazioni di variabile all'esterno di qualsiasi definizione di funzione; esse mantengono i loro valori per tutta l'esecuzione del programma. Alle variabili globali e alle funzioni può fare riferimento qualsiasi funzione definita in seguito alle loro dichiarazioni o definizioni nello stesso file. Questo giustifica l'uso di prototipi di funzioni; quando includiamo **stdio.h** in un programma che chiama **printf**, il prototipo di funzione è posto all'inizio del nostro file per rendere noto il nome **printf** al resto del file.

Definire una variabile globale invece che locale permette che si verifichino effetti secondari non voluti quando una funzione che non ha necessità di accedere alla variabile la modifica accidentalmente o intenzionalmente. In generale, le variabili globali vanno evitate, tranne in certe situazioni con critici requisiti di prestazioni (vedi Capitolo 15). Le variabili usate solo in una data funzione devono essere definite come variabili locali in quella funzione.

Le variabili locali **static** sono ancora conosciute solo nella funzione in cui sono definite e mantengono il loro valore quando si esce dalla funzione. La volta successiva che la funzione è chiamata, la variabile locale **static** contiene il valore che aveva prima che si uscisse dalla funzione la volta precedente. L'istruzione seguente dichiara la variabile locale **count** come **static** e la inizializza a 1:

```
static int count = 1;
```

Tutte le variabili numeriche con permanenza in memoria statica sono inizializzate automaticamente a zero se non vengono inizializzate esplicitamente.

Le parole chiave **extern** e **static** hanno un significato speciale quando sono esplicitamente applicate a identificatori esterni. Nel Capitolo 15 esamineremo l'uso esplicito di **extern** e **static** con identificatori esterni e con programmi con più file sorgente.

### Autovalutazione

1. (*Completare*) Ogni identificatore in un programma ha attributi, quali la classe di memoria, la permanenza in memoria, il \_\_\_\_\_ e il \_\_\_\_\_.

Risposta: campo d'azione, collegamento.

2. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è falsa?

- a) La permanenza in memoria di un identificatore è il periodo durante il quale un identificatore esiste nella memoria.

- b) Il campo d'azione di un identificatore determina dove l'identificatore può essere usato nel programma.
- c) La parola chiave `auto` è usata per dichiarare le variabili con permanenza in memoria automatica. Tali variabili sono create quando il controllo del programma entra nel blocco nel quale sono definite; esse esistono finché il blocco è attivo e sono distrutte quando il controllo del programma esce dal blocco.
- d) Tutte le affermazioni precedenti sono vere.

Risposta: d.

## 5.13 Regole per il campo d'azione

Il campo d'azione di un identificatore è la porzione del programma in cui l'identificatore può essere menzionato. Per esempio, è possibile fare riferimento a una variabile locale in un blocco solo seguendo la sua definizione in quel blocco o in blocchi annidati al suo interno. I quattro campi d'azione per gli identificatori sono il campo d'azione esteso alla funzione, il campo d'azione esteso al file, il campo d'azione esteso al blocco e il campo d'azione esteso al prototipo di funzione.

### Campo d'azione esteso alla funzione

Le etichette sono identificatori seguiti da due punti, come `start:`. Le etichette sono gli *unici* identificatori con il campo d'azione esteso alla funzione. Le etichette possono essere usate ovunque nella funzione in cui compaiono, ma non possono essere menzionate al di fuori del corpo della funzione. Le etichette sono usate nelle istruzioni `switch` (come le etichette `case`) e nelle istruzioni `goto` (vedi Capitolo 15). Le etichette sono nascoste nella funzione in cui sono definite. Questo **occultamento delle informazioni** è un mezzo per implementare il **principio del privilegio minimo**, un principio fondamentale per una buona ingegneria del software. Nel contesto di un'applicazione, il principio afferma che al codice deve essere garantita soltanto la quantità di privilegi e di accessi necessaria per eseguire il suo compito designato, ma non di più.

### Campo d'azione esteso al file

Un identificatore dichiarato al di fuori di una qualsiasi funzione ha un campo d'azione esteso al file. Un tale identificatore è “conosciuto” (cioè accessibile) in tutte le funzioni dal punto in cui l'identificatore è dichiarato fino alla fine del file. Le variabili globali, le definizioni di funzione e i prototipi di funzioni posti al di fuori di una funzione hanno tutti il campo d'azione esteso al file.

### Campo d'azione esteso al blocco

Gli identificatori definiti dentro un blocco hanno il campo d'azione esteso al blocco. Il campo d'azione esteso al blocco termina alla parentesi graffa destra `()` che chiude il blocco. Le variabili locali definite all'inizio di una funzione hanno il campo d'azione esteso al blocco, come anche i parametri della funzione, anch'essi considerati variabili locali dalla funzione. Qualsiasi blocco può contenere definizioni di variabili. Quando i blocchi sono annidati e un identificatore in un blocco esterno ha lo stesso nome di un identificatore in un blocco interno, l'identificatore nel blocco esterno è nascosto finché il blocco interno non termina. Durante l'esecuzione nel blocco interno, il blocco interno vede il valore del proprio identificatore locale, *non* il valore dell'identificatore dal nome identico del blocco in cui è annidato. Per questo motivo, si dovrebbe evitare l'uso di nomi di variabili che nascondono nomi in campi d'azione esterni. Le variabili locali dichiarate `static` hanno ancora il campo d'azione esteso al blocco, anche se esistono da prima che il programma si avvii. Pertanto, la permanenza in memoria *non* si ripercuote sul campo d'azione di un identificatore.

### Campo d'azione esteso al prototipo di funzione

Gli unici identificatori con il campo d'azione esteso al prototipo di funzione sono quelli usati nella lista dei parametri del prototipo di funzione. Come precedentemente accennato, i prototipi di funzioni non richiedono nomi nella lista dei parametri: sono necessari solo i tipi. Se si usa un nome nella lista dei parametri di un prototipo di funzione, il compilatore lo ignora. Gli identificatori usati in un prototipo di funzione possono essere riutilizzati altrove nel programma senza ambiguità.

### Esempio di campo d'azione

Il programma della Figura 5.8 illustra alcuni aspetti relativi ai campi d'azione con variabili globali, variabili locali e variabili locali statiche. Una variabile globale *x* è definita e inizializzata a 1 (riga 9). Questa variabile globale è nascosta in un qualunque blocco (o funzione) in cui è definita una variabile denominata anch'essa *x*. Nella funzione *main*, una variabile locale *x* è definita e inizializzata a 5 (riga 12). Questa variabile viene quindi stampata per mostrare che la *x* globale è nascosta in *main*. In seguito, un nuovo blocco viene definito in *main* con un'altra variabile locale *x* inizializzata a 7 (riga 17). Questa variabile viene stampata per mostrare che essa nasconde *x* nel blocco esterno di *main*. La variabile *x* con valore 7 viene automaticamente eliminata quando si esce dal blocco, dopodiché la variabile locale *x* nel blocco esterno di *main* viene ristampata per mostrare che non è più nascosta.

```

1 // fig05_08.c
2 // Verifica del campo d'azione delle variabili.
3 #include <stdio.h>
4
5 void useLocal(void); // prototipo di funzione
6 void useStaticLocal(void); // prototipo di funzione
7 void useGlobal(void); // prototipo di funzione
8
9 int x = 1; // variabile globale
10
11 int main(void) {
12 int x = 5; // variabile locale per main
13
14 printf("local x in outer scope of main is %d\n", x);
15
16 { // inizio di un nuovo campo d'azione
17 int x = 7; // variabile locale nel nuovo campo d'azione
18
19 printf("local x in inner scope of main is %d\n", x);
20 } // fine del nuovo campo d'azione
21
22 printf("local x in outer scope of main is %d\n", x);
23
24 useLocal(); // useLocal ha una x locale automatica
25 useStaticLocal(); // useStaticLocal ha una x locale statica
26 useGlobal(); // useGlobal usa una x globale
27 useLocal(); // useLocal reinizializza una x locale automatica
28 useStaticLocal(); // la x locale statica conserva il suo valore
29 useGlobal(); // anche la x globale conserva il suo valore
30
31 printf("\nlocal x in main is %d\n", x);
32 }
33
34 // useLocal reinizializza la variabile locale x durante ogni chiamata
35 void useLocal(void) {
36 int x = 25; // inizializzata ogni volta che useLocal e' chiamata
37
38 printf("\nlocal x in useLocal is %d after entering useLocal\n", x);
39 ++x;
40 printf("local x in useLocal is %d before exiting useLocal\n", x);
41 }
42

```

```

43 // useStaticLocal inizializza la variabile locale statica x solo la
44 // prima volta che la funzione e' chiamata; il valore di x e'
45 // conservato tra una chiamata e l'altra a questa funzione
46 void useStaticLocal(void) {
47 static int x = 50; // inizializzata una volta
48
49 printf("\nlocal static x is %d on entering useStaticLocal\n", x);
50 ++x;
51 printf("local static x is %d on exiting useStaticLocal\n", x);
52 }
53
54 // la funzione useGlobal modifica la variabile globale x in ogni chiamata
55 void useGlobal(void) {
56 printf("\nglobal x is %d on entering useGlobal\n", x);
57 x *= 10;
58 printf("global x is %d on exiting useGlobal\n", x);
59 }

```

```

local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5

```

**Figura 5.8** Verifica del campo d'azione delle variabili.

Il programma definisce tre funzioni che non ricevono argomenti e non restituiscono niente. La funzione `useLocal` definisce una variabile locale `x` e la inizializza a 25 (riga 36). Quando la funzione `useLocal` viene chiamata, la variabile viene stampata, incrementata e stampata di nuovo prima dell'uscita dalla funzione. Ogni volta che questa funzione è chiamata, la variabile locale `x` è reinizializzata a 25.

La funzione `useStaticLocal` definisce una variabile `static x` e la inizializza a 50 alla riga 47 (ricordate che la memoria per le variabili `static` è allocata e inizializzata solo una volta prima che il programma inizi l'esecuzione). Le variabili locali dichiarate come `static` mantengono i loro valori anche quando sono fuori dal campo d'azione. Quando `useStaticLocal` è chiamata, `x` viene stampata, incrementata e ristampata prima

dell'uscita dalla funzione. Nella chiamata successiva di questa funzione, la variabile locale `static x` conterrà il valore 51 incrementato precedentemente.

La funzione `useGlobal` non definisce alcuna variabile. Pertanto, quando essa si riferisce alla variabile `x`, viene usata la `x` globale (riga 9). Quando `useGlobal` è chiamata, la variabile globale viene stampata, moltiplicata per 10 e ristampata prima dell'uscita dalla funzione. La volta successiva che la funzione `useGlobal` è chiamata, la variabile globale ha ancora il suo valore modificato, 10. Infine, il programma stampa la variabile locale `x` di nuovo in `main` (riga 31) per mostrare che nessuna delle chiamate di funzione ha modificato il valore di `x` poiché tutte le funzioni si riferiscono a variabili in altri campi d'azione.

### ✓ Autovalutazione

1. (*Completare*) Il \_\_\_\_\_ di un identificatore è la porzione del programma in cui l'identificatore può essere menzionato. Per esempio, è possibile fare riferimento a una variabile locale in un blocco solo seguendo la sua definizione in quel blocco o in blocchi annidati al suo interno.

**Risposta:** campo d'azione.

2. (*Vero/Falso*) Qualsiasi blocco può contenere definizioni di variabili. Quando i blocchi sono annidati e un identificatore di un blocco esterno ha lo stesso nome di un identificatore di un blocco interno, l'identificatore del blocco interno è nascosto finché il blocco esterno non termina.

**Risposta:** *Falso*. In realtà, quando i blocchi sono annidati e un identificatore di un blocco esterno ha lo stesso nome di un identificatore di un blocco interno, l'identificatore del blocco esterno è nascosto finché il blocco interno non termina.

## 5.14 Ricorsione

Per alcuni tipi di problemi è utile avere funzioni che chiamano se stesse. Una **funzione ricorsiva** è una funzione che *chiama se stessa* direttamente o indirettamente attraverso un'altra funzione. La ricorsione è un argomento complesso, esaminato in profondità nei corsi superiori di informatica. In questo paragrafo e nel prossimo sono presentati alcuni semplici esempi di ricorsione. Questo libro contiene un'ampia trattazione della ricorsione, distribuita nel corso dei Capitoli 5–8, 12 e 13. La tabella nel Paragrafo 5.16 riepiloga gli esempi e gli esercizi sulla ricorsione presenti nel libro.

### Casi di base e chiamate ricorsive

Considereremo dapprima la ricorsione dal punto di vista concettuale, esamineremo poi diversi programmi contenenti funzioni ricorsive. Gli approcci ricorsivi per la risoluzione di problemi hanno un numero di elementi in comune. Una funzione ricorsiva è chiamata per risolvere un problema. La funzione "sa" in realtà soltanto come risolvere i casi più semplici, cioè i cosiddetti **casi di base**. Se la funzione è chiamata con un caso di base, restituisce semplicemente un risultato. Se la funzione è chiamata con un problema più complesso, solitamente divide il problema in due parti concettuali:

- una parte che sa come risolvere;
- una parte che non sa come risolvere.

Per rendere la ricorsione fattibile, quest'ultima parte deve somigliare al problema originario ma essere una versione leggermente più semplice o più piccola. Poiché questo nuovo problema somiglia al problema originario, la funzione lancia (chiama) una nuova copia di se stessa per lavorare sul problema più semplice; questo passo è detto **chiamata ricorsiva o passo di ricorsione**. Il passo di ricorsione include anche un'istruzione `return`, perché il suo risultato sarà combinato con la porzione del problema che la funzione sa come risolvere per formare un risultato che sarà restituito alla funzione chiamante originaria.

Il passo di ricorsione viene eseguito mentre la chiamata originaria alla funzione si arresta, in attesa del risultato dal passo di ricorsione. Il passo di ricorsione può produrre molte di più di tali chiamate ricorsive, mentre la funzione continua a dividere in due parti concettuali ogni problema per il quale è chiamata. Perché termini la ricorsione, ogni volta che la funzione chiama se stessa con una versione leggermente più semplice del problema originario, questa sequenza di problemi più semplici deve alla fine *convergere al caso di base*. Quando la fun-

zione riconosce il caso di base, restituisce un risultato alla copia precedente della funzione, e da qui segue una sequenza di restituzioni in cui si ripercorre all'indietro tutta la sequenza delle chiamate fino a che la chiamata originaria della funzione non restituisce infine il risultato finale alla sua funzione chiamante. Come esempio di questi concetti all'opera, scriviamo un programma ricorsivo per eseguire un comune calcolo matematico.

### Calcolo ricorsivo del fattoriale

Il fattoriale di un intero non negativo  $n$ , scritto  $n!$  (pronunciato “ $n$  fattoriale”), è il prodotto

$$n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 1$$

con  $1!$  uguale a 1, e  $0!$  definito come 1. Per esempio,  $5!$  è il prodotto di  $5 * 4 * 3 * 2 * 1$ , che è uguale a 120.

Il fattoriale di un intero, `number`, maggiore o uguale a 0 si può calcolare *iterativamente* (non ricorsivamente) usando un'istruzione `for` come segue:

```
unsigned long long int factorial = 1;
for (int counter = number; counter > 1; --counter)
 factorial *= counter;
```

Si arriva a una definizione *ricorsiva* della funzione fattoriale osservando la seguente relazione:

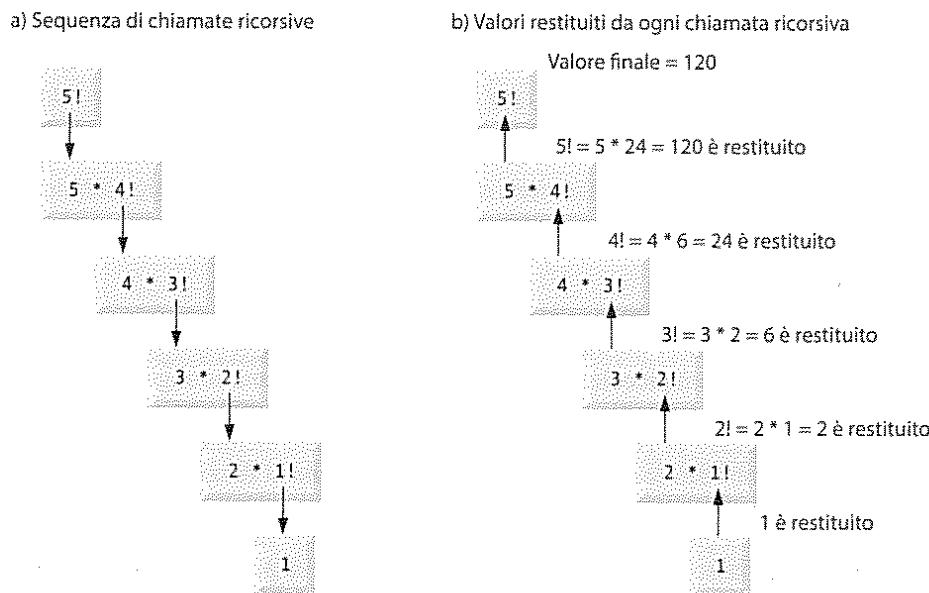
$$n! = n \cdot (n - 1)!$$

Per esempio,  $5!$  è chiaramente uguale a  $5 * 4!$  come mostrato da quanto segue:

$$\begin{aligned} 5! &= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \\ 5! &= 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1) \\ 5! &= 5 \cdot (4!) \end{aligned}$$

### Calcolare $5!$ ricorsivamente

Il calcolo di  $5!$  procede come mostrato nel seguente diagramma. La parte (a) del seguente diagramma mostra come la successione di chiamate ricorsive procede fino a che  $1!$  è calcolato come 1 (cioè il *caso di base*), il che termina la ricorsione. La parte (b) mostra il valore restituito da ogni chiamata ricorsiva alla sua funzione chiamante fino a che è calcolato e restituito il valore finale.



### Implementare calcoli fattoriali con la ricorsione

Il programma della Figura 5.9 usa la ricorsione per calcolare e stampare i fattoriali degli interi da 0 a 21 (la scelta del tipo `unsigned long long int` sarà spiegata fra breve).

```

1 // fig05_09.c
2 // Funzione fattoriale ricorsiva.
3 #include <stdio.h>
4
5 unsigned long long int factorial(int number);
6
7 int main(void) {
8 // calcolo dei fattoriali e stampa del risultato
9 for (int i = 0; i <= 21; ++i) {
10 printf("%d! = %llu\n", i, factorial(i));
11 }
12 }
13
14 // definizione ricorsiva della funzione fattoriale
15 unsigned long long int factorial(int number) {
16 if (number <= 1) { // caso di base
17 return 1;
18 }
19 else { // passo ricorsivo
20 return (number * factorial(number - 1));
21 }
22 }
```

```

0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
```

**Figura 5.9** Funzione fattoriale ricorsiva.

**Funzione factorial**

La funzione ricorsiva `factorial` verifica dapprima se una *condizione di terminazione* è vera, cioè se `number` è minore o uguale a 1. Se `number` è davvero minore o uguale a 1, `factorial` restituisce 1, non è necessaria alcuna ulteriore ricorsione e il programma termina. Se `number` è maggiore di 1, l'istruzione

```
return number * factorial(number - 1);
```

esprime il problema come il prodotto di `number` e di una chiamata ricorsiva a `factorial` che calcola il fattoriale di `number - 1`. La chiamata `factorial(number - 1)` è un problema un po' più semplice dell'originario calcolo `factorial(number)`.

- ⊗ L'omissione del caso di base o la scrittura errata del passo di ricorsione in modo che non converga sul caso di base causerà una ricorsione infinita, con conseguente esaurimento della memoria. Questo è analogo al problema di un ciclo infinito in una soluzione iterativa (non ricorsiva), sebbene i cicli infiniti in genere non esauriscano la memoria.

**I fattoriali diventano grandi velocemente**

La funzione `factorial` (righe 15–22) riceve un `int` e restituisce un risultato di tipo `unsigned long long int`. Il C standard specifica che una variabile di tipo `unsigned long long int` può assumere un valore almeno tanto grande quanto 18.446.744.073.709.551.615. Come si può vedere nella Figura 5.9, i valori fattoriali diventano grandi velocemente. Abbiamo scelto il tipo di dati `unsigned long long int` in modo che il programma possa calcolare valori fattoriali più grandi. La specifica di conversione `%llu` si usa per stampare valori `unsigned long long int`. Purtroppo, la funzione `factorial` produce valori grandi così rapidamente che perfino `unsigned long long int` non ci permette di stampare molti valori fattoriali, poiché il valore massimo di quel tipo viene superato velocemente.

**I tipi interi hanno limiti**

Anche quando usiamo `unsigned long long int`, non riusciamo ancora a calcolare i fattoriali oltre 21! Questo mette in evidenza una debolezza dei linguaggi di programmazione procedurali come il C, nel senso che il linguaggio non può essere facilmente esteso per trattare requisiti specifici di varie applicazioni. I linguaggi orientati agli oggetti come il C++ sono **estensibili**. Con una caratteristica del linguaggio chiamata **classi**, i programmatorei possono creare nuovi tipi di dati, compresi quelli in grado di rappresentare numeri interi arbitrariamente grandi.

**✓ Autovalutazione**

1. **(Completare)** L'omissione del caso di base o la scrittura errata del passo di ricorsione in modo che non converga sul caso di base causerà una \_\_\_\_\_, con conseguente esaurimento della memoria.

**Risposta:** ricorsione infinita.

2. **(Completare)** Il seguente codice dovrebbe calcolare iterativamente il fattoriale di un intero, `number`, ma contiene un errore:

```
unsigned long long int factorial = 1;

for (int counter = number; counter >= 1; --counter)
 factorial * counter;
```

Potete correggere l'errore modificando \_\_\_\_\_ in \_\_\_\_\_.

**Risposta:** \*, \*=.

## 5.15 Esempio di utilizzo della ricorsione: la serie di Fibonacci

La serie di Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

inizia con 0 e 1 e ha la proprietà che ogni successivo numero di Fibonacci è la somma dei due numeri precedenti.

La serie ricorre in natura e, in particolare, descrive una forma a spirale. Il rapporto dei numeri successivi di Fibonacci converge verso un valore costante pari a 1,618... Anche questo numero ricorre ripetutamente in natura ed è stato chiamato *proporzione aurea* o *sezione aurea*. Gli esseri umani sono inclini a trovare la sezione aurea esteticamente attraente. Gli architetti spesso progettano finestre, camere e costruzioni la cui lunghezza e larghezza stanno nel rapporto della sezione aurea. Le cartoline postali sono spesso disegnate con una proporzione lunghezza/larghezza pari alla sezione aurea.

La serie di Fibonacci si può definire ricorsivamente come segue:

```
fibonacci(0) = 0
fibonacci(1) = 1
fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)
```

La Figura 5.10 calcola ricorsivamente l' $n^{\text{mo}}$  numero di Fibonacci usando la funzione fibonacci. I numeri di Fibonacci tendono a diventare grandi velocemente. Pertanto, abbiamo scelto il tipo di dati `unsigned long long int` per il tipo di ritorno nella funzione fibonacci.

```
1 // fig05_10.c
2 // Funzione ricorsiva fibonacci.
3 #include <stdio.h>
4
5 unsigned long long int fibonacci(int n); // prototipo di funzione
6
7 int main(void) {
8 // calcola e stampa fibonacci(number) per 0-10
9 for (int number = 0; number <= 10; number++) {
10 printf("Fibonacci(%d) = %llu\n", number, fibonacci(number));
11 }
12
13 printf("Fibonacci(20) = %llu\n", fibonacci(20));
14 printf("Fibonacci(30) = %llu\n", fibonacci(30));
15 printf("Fibonacci(40) = %llu\n", fibonacci(40));
16 }
17
18 // Definizione ricorsiva della funzione fibonacci
19 unsigned long long int fibonacci(int n) {
20 if (0 == n || 1 == n) { // caso di base
21 return n;
22 }
23 else { // passo ricorsivo
24 return fibonacci(n - 1) + fibonacci(n - 2);
25 }
26 }
```

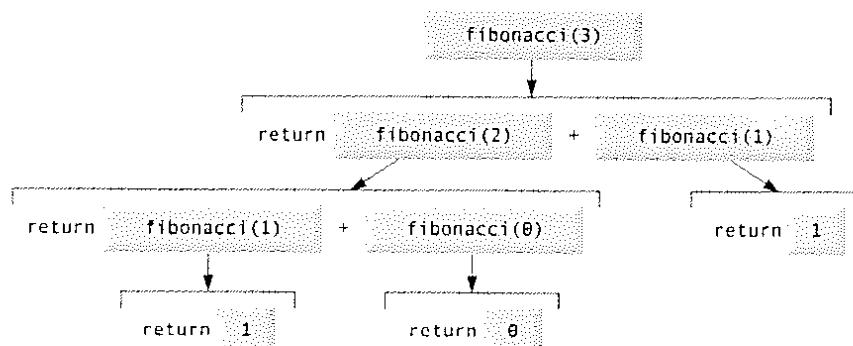
```

Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(3) = 2
Fibonacci(4) = 3
Fibonacci(5) = 5
Fibonacci(6) = 8
Fibonacci(7) = 13
Fibonacci(8) = 21
Fibonacci(9) = 34
Fibonacci(10) = 55
Fibonacci(20) = 6765
Fibonacci(30) = 832040
Fibonacci(40) = 102334155

```

**Figura 5.10** Funzione ricorsiva fibonacci.

Le chiamate a `fibonacci` da `main` *non* sono ricorsive (righe 10 e 13-15), ma tutte le chiamate successive a `fibonacci` sono ricorsive (riga 24). Ogni volta che `fibonacci` è chiamata, essa verifica immediatamente il *caso di base*, se `n` è uguale a 0 o 1. Se questo è vero, viene restituito `n`. È interessante notare che, se `n` è maggiore di 1, il passo di ricorsione genera *due* chiamate ricorsive, ognuna delle quali risolve un problema leggermente più semplice dell'originaria chiamata a `fibonacci`. Il seguente diagramma mostra come la funzione `fibonacci` calcola `fibonacci(3)`:



### Ordine di valutazione degli operandi

Questa figura mette in evidenza alcuni interessanti aspetti riguardanti l'*ordine* in cui i compilatori C valutano gli operandi degli operatori. Questo è un problema diverso dall'ordine in cui gli operatori sono applicati ai loro operandi, ossia l'ordine dettato dalle regole di precedenza e associatività degli operatori. Il diagramma precedente mostra che mentre viene valutata `fibonacci(3)` vengono effettuate *due* chiamate ricorsive, cioè `fibonacci(2)` e `fibonacci(1)`. Ma in quale ordine vengono fatte queste chiamate? Potreste semplicemente supporre che gli operandi vengono valutati da sinistra a destra. Per ragioni di ottimizzazione, il C *non* specifica l'ordine in cui vanno valutati gli operandi della maggior parte degli operatori (incluso `+`). Quindi, non dovete presumere nulla riguardo all'ordine in cui vengono eseguite queste chiamate. Le chiamate potrebbero essere eseguite con `fibonacci(2)` per prima e `fibonacci(1)` dopo, oppure potrebbero essere eseguite in ordine inverso, prima `fibonacci(1)` e poi `fibonacci(2)`. In questo e in molti altri programmi il risultato finale è lo stesso. Ma in alcuni programmi il calcolo di un operando può avere *effetti secondari* che potrebbero incidere sul risultato finale dell'espressione.

### Operatori per i quali l'ordine di valutazione degli operandi è specificato

Il C specifica l'ordine di valutazione degli operandi di soli quattro operatori, ossia `&&`, `||`, l'operatore virgola `(.)` e `? :`. I primi tre di questi sono operatori binari, i cui operandi sono calcolati *da sinistra a destra*. [Nota: le virgole usate per separare gli argomenti nella chiamata di una funzione *non* sono operatori virgola.] L'ultimo operatore è il solo operatore *ternario* del C. Il suo operando più a sinistra viene sempre valutato per primo. Se l'operando più a sinistra ha un valore diverso da zero (vero), successivamente viene valutato l'operando centrale e viene ignorato l'ultimo operando. Se l'operando più a sinistra ha un valore uguale a zero (falso), successivamente viene valutato il terzo operando e viene ignorato quello centrale.

### Complessità esponenziale

Un avvertimento è d'obbligo riguardo ai programmi ricorsivi come quello che usiamo qui per generare i numeri di Fibonacci. Ogni livello di ricorsione nella funzione `fibonacci` ha un effetto di raddoppio sul numero delle chiamate. Il numero delle chiamate ricorsive che saranno eseguite per calcolare l' $n^{\text{mo}}$  numero di Fibonacci è "dell'ordine di  $2^n$ ". Questa situazione sfugge rapidamente di mano. Calcolare solo il 20<sup>mo</sup> numero di Fibonacci richiede un numero dell'ordine di  $2^{20}$  o di circa un milione di chiamate, calcolare il 30<sup>mo</sup> numero di Fibonacci richiede un numero dell'ordine di  $2^{30}$  o di circa un miliardo di chiamate, e così via. Gli informatici chiamano questo fenomeno **complessità esponenziale**. Problemi di questa natura rendono umili persino i computer più potenti del mondo! I problemi di complessità in generale, e la complessità esponenziale in particolare, sono generalmente esaminati in dettaglio nei corsi di informatica di livello superiore riguardanti gli algoritmi.

L'esempio mostrato in questo paragrafo usa una soluzione intuitivamente attraente per calcolare i numeri di Fibonacci, ma esistono approcci migliori. L'Esercizio 5.48 chiede di analizzare più in profondità la ricorsione e di proporre approcci alternativi per implementare l'algoritmo ricorsivo di Fibonacci.

### ✓ Autovalutazione

1. (*Vero/Falso*) Per ragioni di ottimizzazione, il C specifica l'ordine in cui vanno valutati gli operandi della maggior parte degli operatori (incluso `+`).

**Risposta:** *Falso*. Per ragioni di ottimizzazione, il C *non* specifica l'ordine in cui vanno valutati gli operandi della maggior parte degli operatori (incluso `+`). Il C specifica l'ordine di valutazione degli operandi di *soli quattro* operatori, ossia `&&`, `||`, l'operatore virgola `(.)` e `? :`.

2. (*Scelta multipla*) Considerate il codice della Figura 5.10, che implementa una funzione `fibonacci` ricorsiva. Quale delle seguenti affermazioni a), b) o c) è *falsa*?

- a) Tutte le chiamate a `fibonacci` della Figura 5.10 sono ricorsive.
- b) Ogni volta che `fibonacci` è chiamata, essa verifica immediatamente il caso di base, se `n` è uguale a 0 o 1. Se questo è vero, viene restituito `n`.
- c) Se `n` è maggiore di 1, il passo di ricorsione genera *due* chiamate ricorsive, ognuna delle quali risolve un problema leggermente più semplice dell'originaria chiamata a `fibonacci`.
- d) Tutte le affermazioni precedenti sono *vere*.

**Risposta:** a) è *falsa*. In realtà, le chiamate a `fibonacci` da `main` non sono chiamate ricorsive, mentre sono ricorsive tutte le successive chiamate a `fibonacci` (riga 24).

## 5.16 La ricorsione rispetto all'iterazione

Nei paragrafi precedenti abbiamo studiato due funzioni che si possono facilmente implementare o ricorsivamente o iterativamente. In questo paragrafo confrontiamo i due approcci ed esaminiamo perché potreste sceglierne uno rispetto all'altro.

### Caratteristiche comuni di iterazione e ricorsione

- L'iterazione e la ricorsione si basano entrambe su una *struttura di controllo*: l'iterazione usa un'*istruzione di iterazione*; la ricorsione usa un'*istruzione di selezione*.

- L'iterazione e la ricorsione implicano entrambe la *ripetizione*: l'iterazione usa esplicitamente un'*istruzione di iterazione*; la ricorsione attua la ripetizione attraverso *ripetute chiamate di funzione*.
- L'iterazione e la ricorsione richiedono ciascuna un *test di terminazione*: l'iterazione termina quando la *condizione di continuazione del ciclo fallisce*; la ricorsione quando *si incontra un caso di base*.
- L'iterazione controllata da contatore e la ricorsione *procedono gradualmente verso la terminazione*: l'iterazione continua a modificare un contatore finché questo assume un valore che *fa fallire la condizione di continuazione del ciclo*; la ricorsione continua a produrre versioni più semplici del problema originario finché non viene *raggiunto il caso di base*.
- Sia l'iterazione che la ricorsione possono andare avanti *all'infinito*: si ha un *ciclo infinito* con l'iterazione se il test di continuazione del ciclo non diventa *mai* falso; si ha una *ricorsione infinita* se il passo di ricorsione *non* riduce ogni volta il problema in modo che esso *converga al caso di base*. L'iterazione e la ricorsione infinita solitamente si verificano come risultato di errori nella logica di un programma.

### Difetti della ricorsione

La ricorsione ha molti difetti. Essa invoca *ripetutamente* il meccanismo di chiamata di funzione, con conseguente *aggravio di calcolo* (detto *overhead*). Ciò può essere dispendioso sia in termini di tempo di elaborazione sia in termini di spazio di memoria. Ogni chiamata ricorsiva fa sì che sia creata *un'altra copia* della funzione (in realtà solo le variabili della funzione vengono copiate); questo può *consumare una considerevole quantità di memoria*. L'iterazione è eseguita normalmente all'interno di una funzione, così non si ha l'aggravio di calcolo delle ripetute chiamate di funzione e dell'assegnazione extra di memoria. Allora, perché scegliere la ricorsione?

### La ricorsione non è richiesta

Qualunque problema che si può risolvere in maniera ricorsiva si può anche risolvere in maniera iterativa (non ricorsivamente). Un approccio ricorsivo si preferisce normalmente a un approccio iterativo quando rispecchia in maniera più naturale il problema e produce un programma più facile da capire e da correggere. Un'altra ragione per scegliere una soluzione ricorsiva sta nel fatto che una soluzione iterativa potrebbe non essere evidente.

### Esempi di ricorsione ed esercizi presenti nel libro

La maggior parte dei libri di programmazione introduce la ricorsione molto più avanti di quanto abbiamo fatto qui. Crediamo che la ricorsione sia un argomento abbastanza ricco e complesso da meritare di essere introdotto presto, con la distribuzione degli esempi nel resto del libro. La tabella seguente riassume riepiloga gli esempi e gli esercizi di ricorsione presenti nel testo.

| Capitolo          | Esempi ed esercizi di ricorsione                                                                                                                                                                                      |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Capitolo 5</i> | Funzione fattoriale<br>Funzione di Fibonacci<br>Massimo comune divisore<br>Moltiplicare due interi<br>Elevare un intero a una potenza intera<br>Torri di Hanoi<br><b>main ricorsivo</b><br>Visualizzare la ricorsione |
| <i>Capitolo 6</i> | Somma degli elementi di un array<br>Stampare un array<br>Stampare un array all'indietro<br>Stampare una stringa all'indietro<br>Controllare se una stringa è palindroma<br>Valore minimo in un array                  |



| Capitolo           | Esempi ed esercizi di ricorsione                                                                                                                                                                                                                                                                  |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | Ricerca lineare<br>Ricerca binaria<br>Otto regine                                                                                                                                                                                                                                                 |
| <i>Capitolo 7</i>  | Attraversamento di un labirinto                                                                                                                                                                                                                                                                   |
| <i>Capitolo 8</i>  | Stampare all'indietro una stringa in ingresso dalla tastiera                                                                                                                                                                                                                                      |
| <i>Capitolo 12</i> | Ricerca in una lista collegata<br>Stampare all'indietro una lista collegata<br>Inserimento in un albero binario<br>Attraversamento in pre-ordine di un albero binario<br>Attraversamento in ordine di un albero binario<br>Attraversamento in post-ordine di un albero binario<br>Stampare alberi |
| <i>Capitolo 13</i> | Ordinamento per selezione<br>Quicksort (ordinamento veloce)                                                                                                                                                                                                                                       |

### Osservazioni finali

Concludiamo questo capitolo con alcune osservazioni che facciamo ripetutamente in tutto il libro. Una buona ingegneria del software è importante, come lo sono le alte prestazioni, e per questo abbiamo incluso esaurienti suggerimenti per l'ingegneria del software e le prestazioni in tutto il libro. Purtroppo, questi obiettivi sono spesso in contrasto tra loro. Una buona ingegneria del software è la chiave per rendere più fattibile il compito di sviluppare i sistemi software più grandi e più complessi di cui abbiamo bisogno. Le alte prestazioni sono la chiave per realizzare i sistemi del futuro che implementeranno in hardware sempre maggiori funzionalità informatiche. Come entrano in gioco qui le funzioni?

### Ingegneria del software

La suddivisione in funzioni dei programmi di grandi dimensioni favorisce una buona ingegneria del software, ma ha un prezzo. Un programma pesantemente funzionalizzato – confrontato con un programma monolitico (cioè di un solo modulo) senza funzioni – effettua potenzialmente un gran numero di chiamate di funzioni, e ciò consuma tempo d'esecuzione sul processore di un computer. Benché i programmi monolitici possano avere prestazioni migliori, sono più difficili da scrivere, testare, correggere, mantenere e sviluppare.

### Prestazioni

Le odierni architetture hardware sono studiate per rendere efficienti le chiamate di funzione. I compilatori C aiutano a ottimizzare il codice e gli odierni processori hardware e l'architettura multicore sono incredibilmente veloci. Per la maggior parte delle applicazioni e dei sistemi software che costruirete, concentrarsi su una buona ingegneria del software sarà più importante che programmare per ottenere elevate prestazioni. Tuttavia, in molti sistemi e applicazioni, come la programmazione di giochi, i sistemi in tempo reale, i sistemi operativi e i sistemi embedded, le prestazioni sono cruciali; è per questo che nel corso di tutto il libro vi sono consigli riguardanti le prestazioni.

### Autovalutazione

1. (*Vero/Falso*) La suddivisione in funzioni dei programmi di grandi dimensioni favorisce una buona ingegneria del software. Ma un programma pesantemente funzionalizzato – confrontato con un programma monolitico (cioè di un solo modulo) senza funzioni – effettua potenzialmente un gran numero di chiamate di funzioni, e ciò consuma tempo d'esecuzione sul processore di un computer. Benché i programmi monolitici possano avere prestazioni migliori, sono più difficili da scrivere, testare, correggere, mantenere e sviluppare.

**Risposta:** *Vero.*

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) La ricorsione invoca ripetutamente il meccanismo di chiamata di funzione, con conseguente aggravio di calcolo (detto overhead). Ciò può essere dispendioso sia in termini di tempo di elaborazione sia in termini di spazio di memoria.
- b) Ogni chiamata ricorsiva fa sì che venga creata un'altra copia delle istruzioni e delle variabili della funzione; ciò può causare un consumo di memoria considerevole.
- c) L'iterazione è eseguita normalmente all'interno di una funzione, così non si ha l'aggravio di calcolo delle ripetute chiamate di funzione e dell'assegnazione extra di memoria.
- d) Un approccio ricorsivo si preferisce normalmente a un approccio iterativo quando rispecchia in maniera più naturale il problema e produce un programma più facile da capire e da correggere.

**Risposta:** b) è *falsa*. In realtà, ogni chiamata ricorsiva fa sì che venga creata un'altra copia *solo delle variabili della funzione*.



## 5.17 Programmazione sicura in C – Generazione sicura di numeri casuali

Nel Paragrafo 5.10 abbiamo introdotto la funzione `rand` per generare numeri pseudocasuali. Questa funzione è sufficiente per gli esempi dei libri di testo, ma non è pensata per l'uso in applicazioni di livello industriale. Secondo la descrizione della funzione `rand` nel documento del C standard, “Non ci sono garanzie in quanto alla qualità della sequenza casuale prodotta ed è noto che alcune implementazioni producono sequenze con bit meno significativi *incrementalmente non casuali*”. La linea guida del CERT MSC30-C indica che vanno usate le funzioni di generazione di numeri casuali specifiche per l’implementazione per assicurarsi che i numeri casuali prodotti *non siano predibili*. Ciò è estremamente importante, per esempio, nella crittografia e in altre applicazioni di sicurezza.

La linea guida presenta diversi generatori di numeri casuali per specifiche piattaforme che sono considerati sicuri. Per maggiori informazioni, fate riferimento alla linea guida MSC30-C all’indirizzo <https://wiki.sei.cmu.edu>. Se state realizzando applicazioni a livello industriale che richiedono numeri casuali, dovete cercare le funzioni raccomandate per la vostra piattaforma. Per esempio:

- Windows di Microsoft fornisce la funzione `BCryptGenRandom`, che fa parte di “Cryptography API: Next Generation” di Microsoft:  
`https://docs.microsoft.com/en-us/windows/win32/seccng/cng-portal`
- I sistemi basati su POSIX (come Linux) forniscono una funzione `random`, di cui potete ottenere maggiori informazioni eseguendo il comando seguente in una finestra di Terminale o in una shell:  
`man random`
- Il file di intestazione `stdlib.h` di macOS fornisce la funzione `arc4random`, di cui potete ottenere maggiori informazioni eseguendo il comando seguente nella finestra Terminale di macOS:  
`man arc4random`

### ✓ Autovalutazione

1. (*Vero/Falso*) La linea guida del CERT MSC30-C indica che vanno usate le funzioni di generazione di numeri casuali specifiche per l’implementazione per assicurarsi che i numeri casuali prodotti siano predibili. Ciò è estremamente importante, per esempio, nella crittografia e in altre applicazioni di sicurezza.

**Risposta:** *Falso*. In realtà, la linea guida del CERT MSC30-C indica che vanno usate le funzioni di generazione di numeri casuali specifiche per l’implementazione per assicurarsi che i numeri casuali prodotti *non siano predibili*.

## 5.18 Riepilogo

### Paragrafo 5.1 Introduzione

- Il modo migliore per sviluppare e mantenere un programma di grandi dimensioni è quello di **suddividerlo** in diverse parti più piccole, ognuna più maneggevole del programma originario.

### Paragrafo 5.2 Modularizzazione dei programmi in C

- Una **funzione** è invocata con una **chiamata di funzione**, che specifica il nome della funzione e fornisce le informazioni (come argomenti) che occorrono alla funzione chiamata per eseguire il suo compito.

### Paragrafo 5.3 Funzioni della libreria math

- Una funzione viene invocata in un programma scrivendo nell'ordine il nome della funzione, una parentesi sinistra, l'argomento (o una lista di argomenti separati da virgole) e una parentesi destra.
- Ogni argomento può essere una costante, una variabile o un'espressione.

### Paragrafo 5.4 Funzioni

- Vi sono svariate motivazioni per “funzionalizzare” un programma. L'approccio dividi e conquista rende più fattibile lo sviluppo del programma. Un'altra motivazione è la costruzione di nuovi programmi usando funzioni esistenti. Tale riusabilità del software è un concetto chiave nei linguaggi di programmazione orientati agli oggetti derivati dal C, come C++, Java, C#, Objective-C e Swift.
- Con una buona scelta dei nomi e delle definizioni delle funzioni si possono creare programmi partendo da funzioni standardizzate che eseguono compiti specifici, piuttosto che costruirli usando un codice sviluppato personalmente. Ciò è noto come astrazione. Usiamo l'astrazione ogni volta che usiamo funzioni della Libreria Standard come `printf`, `scanf` e `pow`. Una terza motivazione sta nell'evitare di ripetere codice in un programma. Impacchettare il codice come funzione ne permette l'esecuzione in altri punti del programma semplicemente chiamando la funzione.

### Paragrafo 5.5 Definizioni di funzioni

- Gli argomenti passati a una funzione devono concordare in numero, tipo e ordine con i **parametri** nella definizione della funzione.
- Quando un programma incontra la chiamata di una funzione, il controllo è trasferito dal punto di invocazione alla funzione chiamata, vengono eseguite le istruzioni di quella funzione e poi il controllo ritorna alla funzione chiamante.
- Una **funzione chiamata** può restituire il **controllo** a quella chiamante in uno di tre modi. Se la funzione non restituisce un valore, il controllo viene restituito quando si raggiunge la parentesi graffa destra che termina la funzione, oppure eseguendo l'istruzione

```
return;
```

Se la funzione deve restituire un valore, l'istruzione

```
return espressione;
```

restituisce il valore di *espressione*.

- Una **variabile locale** è nota solo in una definizione di funzione. Ad altre funzioni non è consentito conoscere i nomi delle variabili locali di una funzione, né ad alcuna funzione è consentito conoscere i dettagli di implementazione di qualsiasi altra funzione.
- Un **prototipo di funzione** dichiara il nome, il tipo di ritorno e il numero della funzione, i tipi e l'ordine dei parametri che la funzione si aspetta di ricevere.

- Il formato generale per la definizione di una funzione è

```
tipo-del-valore-di-ritorno nome-della-funzione (lista-dei-parametri) {
 istruzioni
}
```

Se una funzione non restituisce un valore, il *tipo-del-valore-di-ritorno* viene dichiarato **void**. Il *nome-della-funzione* è un qualsiasi identificatore valido. La *lista-dei-parametri* è un elenco separato da virgolette contenente le definizioni delle variabili che saranno passate alla funzione. Se una funzione non riceve alcun valore, la *lista-dei-parametri* viene dichiarata **void**.

#### Paragrafo 5.6 Prototipi di funzioni: uno sguardo più approfondito

- I prototipi di funzioni consentono al compilatore di verificare che le funzioni siano chiamate correttamente.
- Il compilatore ignora i nomi delle variabili menzionati nel prototipo di funzione.
- Gli argomenti delle **normali regole di conversione aritmetica** del C standard in un'**espressione con tipi misti** sono convertiti nello stesso tipo.

#### Paragrafo 5.7 Pila delle chiamate delle funzioni e record di attivazione

- Le pile sono note come strutture di dati **last-in, first-out (LIFO)**, ossia l'ultimo elemento inserito nella pila è il primo elemento a essere rimosso da essa.
- Una funzione chiamata deve sapere come tornare alla sua funzione chiamante, così, quando la funzione è chiamata, l'indirizzo di ritorno della funzione chiamante è inserito con un push in cima alla **pila di esecuzione del programma**. Se si ha una sequenza di chiamate di funzioni, gli indirizzi di ritorno successivi vengono inseriti nell'ordine last-in, first-out nella pila, in modo che l'ultima funzione chiamata sia la prima a tornare alla sua funzione chiamante.
- La pila di esecuzione del programma contiene la **memoria per le variabili locali** usate in ciascuna invocazione di funzione durante l'esecuzione del programma. Questa collezione di dati è nota come **record di attivazione** della chiamata della funzione. Quando si effettua una chiamata di una funzione, il record di attivazione per la chiamata di quella funzione è inserito in cima alla pila di esecuzione del programma. Quando la funzione torna alla sua funzione chiamante, il record di attivazione è rimosso dalla cima della pila e le sue variabili locali non sono più note al programma.
- Se vi sono più chiamate di funzione di quanti record di attivazione possono essere memorizzati nella pila di esecuzione del programma, si verifica un errore noto come **overflow della pila**.

#### Paragrafo 5.8 File di intestazione

- Ogni libreria standard ha un corrispondente **file di intestazione** contenente i prototipi di funzioni per tutte le funzioni in quella libreria.
- Potete creare e includere i vostri file di intestazione.

#### Paragrafo 5.9 Passare gli argomenti per valore e per riferimento

- Quando un argomento è **passato per valore**, una copia del suo valore viene creata e passata alla funzione chiamata. Le modifiche alla copia non incidono sul valore della variabile originaria nella funzione chiamante.
- Quando un argomento è **passato per riferimento**, la funzione chiamante consente alla funzione chiamata di modificare il valore della variabile originaria.
- Tutte le chiamate in C sono passate per valore come impostazione predefinita.

### Paragrafo 5.10 Generazione di numeri casuali

- La **funzione rand** genera un intero tra 0 e RAND\_MAX che è definito dal C standard con un valore pari ad almeno 32.767.
- I valori prodotti da **rand** possono essere **scalati e spostati** per produrre valori in un intervallo specifico.
- Per **randomizzare** un programma, usate la funzione della Libreria Standard del C **srand**.
- La **funzione srand** fornisce un seme al generatore di numeri casuali. Una chiamata di **srand** viene solitamente inserita in un programma solo dopo che ne è stato effettuato il debugging completo. Questo assicura la ripetibilità, che è essenziale per provare che le correzioni a un programma di generazione di numeri casuali operino bene.
- I prototipi di funzioni per **rand** e **srand** sono contenuti in <stdlib.h>.
- Per randomizzare un programma senza la necessità di inserire ogni volta un seme, utilizziamo **srand(time(NULL))**.
- L'equazione generale per scalare e spostare di intervallo un numero casuale è

```
int n = a + rand() % b;
```

dove **a** è il valore di spostamento (cioè il primo numero nell'intervallo di interi consecutivi desiderato) e **b** è il fattore di scala (cioè l'ampiezza dell'intervallo di interi consecutivi desiderato).

### Paragrafo 5.11 Caso pratico sulla simulazione di numeri casuali: creare un gioco da casinò

- Un'**enumerazione**, introdotta dalla parola chiave **enum**, è un insieme di costanti intere. I valori in un **enum** partono da 0 e sono incrementati di 1. È anche possibile assegnare un valore intero a ciascun identificatore in un **enum**. Gli identificatori in un'**enumerazione** devono essere unici, ma i valori si possono duplicare.

### Paragrafo 5.12 Classi di memoria

- Ogni identificatore in un programma ha gli attributi **classe di memoria**, **permanenza in memoria**, **campo d'azione** e **collegamento**.
- Il C fornisce quattro **classi di memoria** indicate dagli **specificatori della classe di memoria**: **auto**, **register**, **extern** e **static**.
- La **permanenza in memoria** di un identificatore è il periodo in cui quell'identificatore esiste nella memoria.
- Il **collegamento** di un identificatore determina per un programma con più file sorgente se l'identificatore è conosciuto soltanto nel file sorgente corrente oppure in qualunque file sorgente con le opportune dichiarazioni.
- Le variabili locali di una funzione hanno **permanenza in memoria automatica**: vengono create quando il controllo del programma entra nel blocco in cui sono definite, esistono finché il blocco è attivo e sono distrutte quando il controllo del programma esce dal blocco.
- Le parole chiave **extern** e **static** si usano per dichiarare gli identificatori per le variabili e le funzioni con permanenza in memoria statica. Le variabili con **permanenza in memoria statica** sono allocate e inizializzate una volta soltanto, prima che il programma inizi l'esecuzione.
- Vi sono due tipi di **identificatori con permanenza in memoria statica**: gli **identificatori esterni** (come le variabili globali e i nomi delle funzioni) e le **variabili locali dichiarate con lo specificatore della classe di memoria static**.
- Le **variabili globali** sono create mettendo le definizioni di variabile al di fuori di una qualunque definizione di funzione. Le variabili globali mantengono i loro valori per tutta l'esecuzione del programma.
- Le **variabili locali static** mantengono il loro valore tra le chiamate alla funzione nella quale sono definite.

- Come impostazione predefinita, tutte le variabili numeriche con permanenza in memoria statica vengono inizializzate a zero.

#### Paragrafo 5.13 Regole per il campo d'azione

- Il **campo d'azione** di un identificatore è l'ambito in cui è possibile fare riferimento all'identificatore in un programma.
- Lo scopo dell'**occultamento delle informazioni** fa sì che le funzioni possano accedere solo alle informazioni di cui hanno bisogno per completare le proprie attività. Questo è un mezzo per implementare il **principio del privilegio minimo**.
- Un identificatore può avere il **campo d'azione esteso alla funzione**, il **campo d'azione esteso al file**, il **campo d'azione esteso al blocco** o il **campo d'azione esteso al prototipo di funzione**.
- Le **etichette** sono gli unici identificatori con campo d'azione esteso alla funzione. Le etichette possono essere usate ovunque nella funzione in cui compaiono, ma non è possibile fare riferimento a esse al di fuori del corpo della funzione.
- Un identificatore dichiarato al di fuori di una qualunque funzione ha il campo d'azione esteso al file. Un tale identificatore è “conosciuto” in tutte le funzioni dal punto in cui è dichiarato fino alla fine del file.
- Gli identificatori definiti all'interno di un blocco hanno il campo d'azione esteso al blocco, che termina in corrispondenza della parentesi graffa destra () del blocco.
- Le variabili locali hanno il campo d'azione esteso al blocco, così come i parametri di funzione, che sono variabili locali.
- Un blocco può contenere definizioni di variabili. Quando i blocchi sono annidati e un identificatore in un blocco esterno ha lo stesso nome di un identificatore in un blocco interno, l'identificatore nel blocco esterno è “nascosto” finché il blocco interno non termina.
- Gli unici identificatori con il campo d'azione esteso al prototipo di funzione sono quelli usati nella lista di parametri del prototipo di una funzione.

#### Paragrafo 5.14 Ricorsione

- Una **funzione ricorsiva** è una funzione che chiama se stessa direttamente o indirettamente.
- Se una funzione ricorsiva è chiamata con un **caso di base**, restituisce semplicemente un risultato. Se è chiamata per risolvere un problema più complesso, divide il problema in due parti concettuali: una parte che sa come trattare e una versione leggermente più semplice del problema originario. Poiché questo nuovo problema somiglia al problema originario, la funzione lancia una chiamata ricorsiva per operare sul problema più semplice.
- Perché termini la ricorsione, ogni volta che la funzione ricorsiva chiama se stessa con una versione un po' più semplice del problema originario, la sequenza di problemi sempre più semplici deve convergere al **caso di base**. Quando la funzione riconosce il caso di base, il risultato è restituito alla precedente chiamata della funzione e ciò attiva una sequenza di restituzioni a ritroso, finché la chiamata originaria della funzione restituisce infine il risultato finale.
- Il C standard non specifica l'ordine in cui vanno calcolati gli operandi della maggior parte degli operatori (compreso +). Dei molti operatori del C, lo standard specifica l'ordine di calcolo degli operandi dei soli operatori &&, ||, l'operatore virgola (,) e ?: . I primi tre di questi sono operatori binari, i cui due operandi sono calcolati da sinistra a destra. L'ultimo operatore è l'unico operatore ternario del C. Il suo operando più a sinistra è calcolato per primo; se esso ha valore diverso da zero, viene valutato l'operando centrale e l'ultimo operando viene ignorato; se l'operando più a sinistra ha valore uguale a zero, successivamente viene valutato il terzo operando e l'operando centrale viene ignorato.

#### Paragrafo 5.16 La ricorsione rispetto all'iterazione

- L'iterazione e la ricorsione si basano entrambe su una struttura di controllo: l'iterazione usa un'istruzione di iterazione; la ricorsione usa un'istruzione di selezione.

- L'iterazione e la ricorsione implicano entrambe la ripetizione: l'iterazione usa esplicitamente un'istruzione di iterazione; la ricorsione attua la ripetizione attraverso ripetute chiamate di funzione.
- L'iterazione e la ricorsione richiedono ciascuna un test di terminazione: l'iterazione termina quando la condizione di continuazione del ciclo fallisce; la ricorsione termina quando si incontra un caso di base.
- Sia l'iterazione che la ricorsione possono andare avanti all'infinito: si ha un ciclo infinito con l'iterazione se il test di continuazione del ciclo non diventa mai falso; si ha una ricorsione infinita se il passo di ricorsione non riduce ogni volta il problema in modo che esso converga al caso di base.
- La ricorsione invoca ripetutamente il meccanismo di chiamata di funzione e, conseguentemente, introduce l'aggravio di calcolo (overhead) che questo comporta. Ciò può essere dispendioso sia in termini di tempo del processore sia in termini di spazio di memoria.

## Esercizi di autovalutazione

5.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- a) Per modularizzare i programmi si utilizzano \_\_\_\_\_.
- b) Una funzione è invocata con una \_\_\_\_\_.
- c) Una variabile che è conosciuta solo all'interno della funzione in cui è definita è chiamata \_\_\_\_\_.
- d) L'istruzione \_\_\_\_\_ viene usata per restituire il valore di un'espressione alla funzione chiamante.
- e) La parola chiave \_\_\_\_\_ si usa nell'intestazione di una funzione per indicare che una funzione non restituisce un valore o che una funzione non contiene parametri.
- f) Il \_\_\_\_\_ di un identificatore è la porzione di programma in cui si può usare l'identificatore.
- g) I tre modi di restituire il controllo da una funzione chiamata a una chiamante sono \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.
- h) Un \_\_\_\_\_ consente al compilatore di controllare il numero, i tipi e l'ordine degli argomenti passati a una funzione.
- i) La funzione \_\_\_\_\_ si usa per produrre numeri casuali.
- j) La funzione \_\_\_\_\_ si usa per impostare il seme per i numeri casuali in modo da rendere casuale il programma.
- k) Gli specificatori della classe di memoria sono \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.
- l) Le variabili dichiarate in un blocco o nella lista dei parametri di una funzione sono della classe di memoria \_\_\_\_\_, a meno che non sia specificato diversamente.
- m) Una variabile `non-static` definita al di fuori di un blocco o di una funzione è una variabile \_\_\_\_\_.
- n) Perché una variabile locale in una funzione mantenga il suo valore tra le chiamate alla funzione deve essere dichiarata con lo specificatore della classe di memoria \_\_\_\_\_.
- o) Le quattro possibili estensioni del campo d'azione di un identificatore sono \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.
- p) Una funzione che chiama se stessa direttamente o indirettamente è una funzione \_\_\_\_\_.
- q) Una funzione ricorsiva ha tipicamente due componenti: una che fornisce una modalità perché la ricorsione termini verificando un caso \_\_\_\_\_ e una che esprime il problema come una chiamata ricorsiva per un problema leggermente più semplice della chiamata originaria.

5.2 Considerate il seguente programma.

```

1 #include <stdio.h>
2 int cube(int y);
3
4 int main(void) {
5 for (int x = 1; x <= 10; ++x) {
6 printf("%d\n", cube(x));
7 }
8 }
9
10 int cube(int y) {

```

```

11 return y * y * y;
12 }
```

Determinate l'estensione del campo d'azione (alla funzione, al file, al blocco o al prototipo di funzione) di ognuno dei seguenti elementi.

- a) La variabile `x` in `main`.
- b) La variabile `y` in `cube`.
- c) La funzione `cube`.
- d) La funzione `main`.
- e) Il prototipo di funzione per `cube`.
- f) L'identificatore `y` nel prototipo di funzione per `cube`.

**5.3** Scrivete un programma che verifichi se gli esempi delle chiamate a funzioni della libreria `math` mostrati nella tabella del Paragrafo 5.3 producano realmente i risultati indicati.

**5.4** Assegnate l'intestazione di funzione a ognuna delle seguenti funzioni.

- a) La funzione `hypotenuse` che riceve due argomenti `double`, `side1` e `side2`, e restituisce un risultato `double`.
- b) La funzione `smallest` che riceve tre interi, `x`, `y`, `z`, e restituisce un intero.
- c) La funzione `instructions` che non riceve alcun argomento e non restituisce alcun valore.
- d) La funzione `intToFloat` che riceve un argomento intero, `number`, e restituisce un `float`.

**5.5** Scrivete il prototipo di funzione per ognuna delle seguenti funzioni:

- a) La funzione descritta nell'Esercizio 5.4(a).
- b) La funzione descritta nell'Esercizio 5.4(b).
- c) La funzione descritta nell'Esercizio 5.4(c).
- d) La funzione descritta nell'Esercizio 5.4(d).

**5.6** Scrivete una dichiarazione per la variabile in virgola mobile `lastValue` che deve mantenere il suo valore tra le chiamate alla funzione in cui è definita.

**5.7** Trovate l'errore in ognuno dei seguenti segmenti di programma e spiegate come può essere corretto (vedi anche Esercizio 5.46):

- a) `int g(void) {`  
     `printf("%s", "Inside function g\n");`  
     `int h(void) {`  
         `printf("%s", "Inside function h\n");`  
     `}`  
    `}`
- b) `int sum(int x, int y) {`  
     `int result = x + y;`  
    `}`
- c) `void f(float a); {`  
     `float a;`  
     `printf("%f", a);`  
    `}`
- d) `int sum(int n) {`  
     `if (0 == n) {`  
         `return 0;`  
     `}`  
     `else {`  
         `n + sum(n - 1);`  
     `}`  
    `}`

```

e) void product(void) {
 printf("%s", "Enter three integers: ")
 int a;
 int b;
 int c;
 scanf("%d%d%d", &a, &b, &c);
 int result = a * b * c;
 printf("Result is %d", result);
 return result;
}

```

### Risposte agli esercizi di autovalutazione

**5.1** a) funzioni. b) chiamata di funzione. c) variabile locale. d) `return`. e) `void`. f) campo d'azione. g) `return`, `return expression` oppure incontrando la parentesi graffa destra che chiude la funzione. h) prototipo di funzione. i) `rand`. j) `srand`. k) `auto`, `register`, `extern`, `static`. l) `auto`. m) esterna, globale. n) `static`. o) la funzione, il file, il blocco, il prototipo di funzione. p) ricorsiva. q) di base.

**5.2** a) il blocco. b) il blocco. c) il file. d) il file. e) il file. f) il prototipo di funzione.

**5.3** Si veda sotto. [Nota: sulla maggior parte dei sistemi Linux, quando compilate questo programma dovete usare l'opzione `-lm`.]

```

1 // ex05_03.c
2 // Verifica delle funzioni della libreria math
3 #include <stdio.h>
4 #include <math.h>
5
6 int main(void) {
7 // calcola e stampa la radice quadrata
8 printf("sqrt(%.1f) = %.1f\n", 900.0, sqrt(900.0));
9 printf("sqrt(%.1f) = %.1f\n", 9.0, sqrt(9.0));
10
11 // calcola e stampa la radice cubica
12 printf("cbrt(%.1f) = %.1f\n", 27.0, cbrt(27.0));
13 printf("cbrt(%.1f) = %.1f\n", -8.0, cbrt(-8.0));
14
15 // calcola e stampa la funzione esponenziale
16 printf("exp(%.1f) = %f\n", 1.0, exp(1.0));
17 printf("exp(%.1f) = %f\n", 2.0, exp(2.0));
18
19 // calcola e stampa il logaritmo (in base e)
20 printf("log(%f) = %.1f\n", 2.718282, log(2.718282));
21 printf("log(%f) = %.1f\n", 7.389056, log(7.389056));
22
23 // calcola e stampa il logaritmo (in base 10)
24 printf("log10(%.1f) = %.1f\n", 1.0, log10(1.0));
25 printf("log10(%.1f) = %.1f\n", 10.0, log10(10.0));
26 printf("log10(%.1f) = %.1f\n", 100.0, log10(100.0));
27
28 // calcola e stampa il valore assoluto
29 printf("fabs(%.1f) = %.1f\n", 13.5, fabs(13.5));
30 printf("fabs(%.1f) = %.1f\n", 0.0, fabs(0.0));
31 printf("fabs(%.1f) = %.1f\n", -13.5, fabs(-13.5));
32

```

```

33 // calcola e stampa ceil(x)
34 printf("ceil(%.1f) = %.1f\n", 9.2, ceil(9.2));
35 printf("ceil(%.1f) = %.1f\n", -9.8, ceil(-9.8));
36
37 // calcola e stampa floor(x)
38 printf("floor(%.1f) = %.1f\n", 9.2, floor(9.2));
39 printf("floor(%.1f) = %.1f\n", -9.8, floor(-9.8));
40
41 // calcola e stampa pow(x, y)
42 printf("pow(%.1f, %.1f) = %.1f\n", 2.0, 7.0, pow(2.0, 7.0));
43 printf("pow(%.1f, %.1f) = %.1f\n", 9.0, 0.5, pow(9.0, 0.5));
44
45 // calcola e stampa fmod(x, y)
46 printf("fmod(%.3f, %.3f) = %.3f\n", 13.657, 2.333,
47 fmod(13.657, 2.333));
48
49 // calcola e stampa sin(x)
50 printf("sin(%.1f) = %.1f\n", 0.0, sin(0.0));
51
52 // calcola e stampa cos(x)
53 printf("cos(%.1f) = %.1f\n", 0.0, cos(0.0));
54
55 // calcola e stampa tan(x)
56 printf("tan(%.1f) = %.1f\n", 0.0, tan(0.0));
57 }

```

```

sqrt(900.0) = 30.0
sqrt(9.0) = 3.0
cbrt(27.0) = 3.0
cbrt(-8.0) = -2.0
exp(1.0) = 2.718282
exp(2.0) = 7.389056
log(2.718282) = 1.0
log(7.389056) = 2.0
log10(1.0) = 0.0
log10(10.0) = 1.0
log10(100.0) = 2.0
fabs(13.5) = 13.5
fabs(0.0) = 0.0
fabs(-13.5) = 13.5
ceil(9.2) = 10.0
ceil(-9.8) = -9.0
floor(9.2) = 9.0
floor(-9.8) = -10.0
pow(2.0, 7.0) = 128.0
pow(9.0, 0.5) = 3.0
fmod(13.657, 2.333) = 1.992
sin(0.0) = 0.0
cos(0.0) = 1.0
tan(0.0) = 0.0

```

- 5.4 a) `double hypotenuse(double side1, double side2)`  
      b) `int smallest(int x, int y, int z)`  
      c) `void instructions(void)`  
      d) `float intToFloat(int number)`
- 5.5 a) `double hypotenuse(double side1, double side2);`  
      b) `int smallest(int x, int y, int z);`  
      c) `void instructions(void);`  
      d) `float intToFloat(int number);`
- 5.6 `static float lastValue;`
- 5.7 a) Errore: la funzione `h` è definita nella funzione `g`.  
     Correzione: spostate la definizione di `h` fuori dalla definizione di `g`.  
     b) Errore: il corpo della funzione dovrebbe restituire un intero, ma non lo fa.  
     Correzione: sostituite l'istruzione nel corpo della funzione con:  
`return x + y;`  
     c) Errore: il punto e virgola dopo la parentesi destra che chiude la lista dei parametri e la ridefinizione del parametro `a` nella definizione di funzione.  
     Correzione: cancellate il punto e virgola dopo la parentesi destra della lista dei parametri e cancellate la dichiarazione `float a;` nel corpo della funzione.  
     d) Errore: il risultato di `n + sum(n - 1)` non viene restituito; `sum` restituisce un risultato scorretto.  
     Correzione: riscrivete l'istruzione nella clausola `else` come  
`return n + sum(n - 1);`  
     e) Errore: la funzione restituisce un valore, mentre non deve farlo.  
     Correzione: eliminate l'istruzione `return`.

## Esercizi

- 5.8 Determinate il valore di `x` dopo l'esecuzione di ciascuna delle seguenti istruzioni.
- a) `x = fabs(7.5);`  
   b) `x = floor(7.5);`  
   c) `x = fabs(0.0);`  
   d) `x = ceil(0.0);`  
   e) `x = fabs(-6.4);`  
   f) `x = ceil(-6.4);`  
   g) `x = ceil(-fabs(-8 + floor(-5.5)));`
- 5.9 (*Costo del parcheggio*) Un garage fa pagare una tariffa minima di \$2,00 per parcheggiare fino a tre ore, più \$0,50 all'ora per ogni ora *o parte di essa* oltre le tre ore. Il costo massimo per un periodo di 24 ore è di \$10,00. Supponete che nessuna macchina resti parcheggiata per più di 24 ore. Scrivete un programma che calcoli e stampi i costi del parcheggio per ciascuno dei tre clienti che ieri hanno parcheggiato le loro auto in questo garage. Dovete inserire le ore di parcheggio per ogni cliente. Il vostro programma deve stampare i risultati in un formato tabellare e deve calcolare e stampare il totale degli incassi di ieri. Il programma deve usare la funzione `calculateCharges` per determinare il costo per ogni cliente. Il vostro output deve avere il seguente formato:

| Car          | Hours       | Charge       |
|--------------|-------------|--------------|
| 1            | 1.5         | 2.00         |
| 2            | 4.0         | 2.50         |
| 3            | 24.0        | 10.00        |
| <b>TOTAL</b> | <b>29.5</b> | <b>14.50</b> |

**5.10 (Arrotondamento)** Un'applicazione della funzione `floor` è l'arrotondamento di un valore all'intero più vicino. L'istruzione

```
y = floor(x + .5);
```

arrotonda il numero  $x$  all'intero più vicino e assegna il risultato a  $y$ . Scrivete un programma che legga diversi numeri e arrotondi ciascuno di essi all'intero più vicino. Per ogni numero processato stampate sia il numero originario sia il numero arrotondato.

**5.11 (Arrotondamento)** La funzione `floor` può essere usata per arrotondare un numero in riferimento a una specifica posizione decimale. L'istruzione

```
y = floor(x * 10 + .5) / 10;
```

arrotonda  $x$  alla posizione dei decimi (la prima posizione alla destra del punto decimale). L'istruzione

```
y = floor(x * 100 + .5) / 100;
```

arrotonda  $x$  alla posizione dei centesimi (la seconda posizione alla destra del punto decimale). Scrivete un programma che definisca quattro funzioni per arrotondare un numero  $x$  in vari modi.

- a) `roundToInteger(number)`
- b) `roundToTenths(number)`
- c) `roundToHundredths(number)`
- d) `roundToThousandths(number)`

Per ogni valore letto, il vostro programma deve stampare il valore originario, il numero arrotondato all'intero più vicino, il numero arrotondato alla posizione dei decimi, il numero arrotondato alla posizione dei centesimi e il numero arrotondato alla posizione delle migliaia.

**5.12** Rispondete a ognuna delle seguenti domande:

- a) Cosa significa scegliere numeri "a caso"?
- b) Perché la funzione `rand` è utile per simulare i giochi d'azzardo?
- c) Perché randomizzereste un programma usando `srand`? In quali circostanze è desiderabile non randomizzare?
- d) Perché spesso è necessario scalare e/o spostare di intervallo i valori prodotti da `rand`?

**5.13** Scrivete istruzioni che assegnino valori interi casuali alla variabile  $n$  nei seguenti intervalli.

- a)  $1 \leq n \leq 2$
- b)  $1 \leq n \leq 100$
- c)  $0 \leq n \leq 9$
- d)  $1000 \leq n \leq 1112$
- e)  $-1 \leq n \leq 1$
- f)  $-3 \leq n \leq 11$

**5.14** Per ognuno dei seguenti insiemi di interi scrivete un'istruzione singola per stampare un numero tratto a caso dall'insieme.

- a) 2, 4, 6, 8, 10.
- b) 3, 5, 7, 9, 11.
- c) 6, 10, 14, 18, 22.

**5.15 (Calcolo dell'ipotenusa)** Definite una funzione chiamata `hypotenuse` che calcoli la lunghezza dell'ipotenusa di un triangolo rettangolo quando sono dati gli altri due lati. La funzione deve ricevere due argomenti di tipo `double` e restituire l'ipotenusa come un `double`. Testate il vostro programma con i valori dei lati specificati nella seguente tabella:

| Lato 1 | Lato 2 |
|--------|--------|
| 3.0    | 4.0    |
| 5.0    | 12.0   |
| 8.0    | 15.0   |

- 5.16 (Esponenziazione)** Scrivete una funzione `integerPower(base, exponent)` che restituisca il valore di  $\text{base}^{\text{exponent}}$

Per esempio, `integerPower(3, 4) = 3 * 3 * 3 * 3`. Supponete che `exponent` sia un intero positivo diverso da zero e che `base` sia un intero. La funzione `integerPower` deve usare un'istruzione `for` per controllare il calcolo. Non usate alcuna funzione della libreria `math`.

- 5.17 (Multipli)** Scrivete una funzione `isMultiple` che determini per una coppia di interi se il secondo intero sia un multiplo del primo. La funzione deve ricevere due argomenti interi e restituire 1 (vero) se il secondo è un multiplo del primo e 0 (falso) nel caso contrario. Usate questa funzione in un programma che riceve in ingresso una serie di coppie di interi.

- 5.18 (Pari o dispari)** Scrivete un programma che riceva in ingresso una serie di interi e li passi uno alla volta alla funzione `isEven`, che usa l'operatore di resto per determinare se un intero è pari. La funzione deve prendere un argomento intero e restituire 1 se l'intero è pari e 0 nel caso contrario.

- 5.19 (Quadrato di asterischi)** Scrivete una funzione che stampi un quadrato di asterischi pieno il cui lato è specificato nel parametro intero `side`. Per esempio, se `side` è 4, la funzione stampa:

```



```

- 5.20 (Stampare un quadrato di un qualunque carattere)** Modificate la funzione realizzata nell'Esercizio 5.19 per formare il quadrato con qualsiasi carattere contenuto nel parametro `fillCharacter` di tipo `char`. Così, se `side` è 5 e `fillCharacter` è "#", questa funzione deve stampare:

```
#####
#####
#####
#####
#####
```

- 5.21 (Progetto: disegnare forme e caratteri)** Usate tecniche simili a quelle sviluppate negli Esercizi 5.19 e 5.20 per produrre un programma che disegni una vasta gamma di forme.

- 5.22 (Separazione di cifre)** Scrivete segmenti di programma che effettuino le seguenti operazioni:

- Calcolo della parte `int` del quoziente quando `int a` è diviso per `int b`.
- Calcolo del resto `int` quando `int a` è diviso per `int b`.
- Usate le parti di programma sviluppate in a) e b) per scrivere una funzione che riceva in ingresso un intero tra 1 e 32767 e lo stampi come una sequenza di cifre, con due spazi tra ognuna di esse. Per esempio, 4562 deve essere stampato come:

```
4 5 6 2
```

- 5.23 (Tempo in secondi)** Scrivete una funzione che riceva il tempo espresso come tre argomenti interi (per ore, minuti e secondi) e restituisca il numero dei secondi da quando l'orologio "ha battuto le 12" l'ultima volta. Usate questa funzione per calcolare la quantità di tempo in secondi tra due orari, entrambi contenuti entro un ciclo dell'orologio di 12 ore.

- 5.24 (Conversioni di temperatura)** Implementate le seguenti funzioni intere.

- `toCelsius` restituisce l'equivalente in gradi Celsius di una temperatura in gradi Fahrenheit.
- `toFahrenheit` restituisce l'equivalente in gradi Fahrenheit di una temperatura in gradi Celsius.

Usate queste funzioni per scrivere un programma che stampi diagrammi che mostrino gli equivalenti gradi Fahrenheit di tutte le temperature in gradi Celsius da 0 a 100 gradi e gli equivalenti gradi Celsius di tutte le temperature in gradi Fahrenheit da 32 a 212 gradi. Stampate gli output in un formato tabellare che minimizzi il numero delle righe di output, pur rimanendo ancora leggibile.

**5.25 (*Trovare il minimo*)** Scrivete una funzione che restituisca il più piccolo di tre numeri in virgola mobile.

**5.26 (*Numeri perfetti*)** Un numero intero è un *numero perfetto* se i suoi fattori, compreso 1 (ma non il numero stesso), hanno come somma il numero stesso. Per esempio, 6 è un numero perfetto perché  $6 = 1 + 2 + 3$ . Scrivete una funzione `isPerfect` che determini se il parametro `number` è un numero perfetto. Usate questa funzione in un programma che determini e stampi tutti i numeri perfetti tra 1 e 1.000. Stampate i fattori di ogni numero perfetto per confermare che il numero è effettivamente perfetto. Sfivate la potenza del vostro computer provando numeri molto più grandi di 1.000.

**5.27 (*Numeri primi*)** Un intero è un numero *primo* se è divisibile solo per 1 e per se stesso. Per esempio, 2, 3, 5 e 7 sono numeri primi, ma 4, 6, 8 e 9 non lo sono. Scrivete una funzione che determini se un numero è primo. Usate questa funzione in un programma che determini e stampi tutti i numeri primi tra 1 e 10.000. Quanti di questi 10.000 numeri dovete realmente provare prima di essere sicuri di aver trovato tutti i numeri primi? Inizialmente potreste pensare che  $n/2$  è il limite superiore dei test per verificare se un numero è primo, ma in realtà dovete solo spingervi sino alla radice quadrata di  $n$ . Riscrivete il programma e fatelo eseguire in tutti e due i modi. Valutate il miglioramento delle prestazioni.

**5.28 (*Inversione di cifre*)** Scrivete una funzione che riceva un valore intero e restituisca il numero con le sue cifre invertite. Per esempio, dato il numero 7631, la funzione deve restituire 1367.

**5.29 (*Massimo comun divisore*)** Il *massimo comun divisore* (GCD, Greatest Common Divisor) di due interi è l'intero più grande che divide in parti uguali ognuno dei due numeri. Scrivete la funzione `gcd` che restituisce il massimo comun divisore di due interi.

**5.30 (*Valutazione qualitativa dei voti di uno studente*)** Scrivete una funzione `toQualityPoints` che riceva in ingresso la media dei voti di uno studente e restituisca 4 se questa è compresa nell'intervallo 90–100, 3 se è tra 80–89, 2 se è tra 70–79, 1 se è tra 60–69 e 0 se la media è più bassa di 60.

**5.31 (*Lancio di una moneta*)** Scrivete un programma che simuli il lancio di una moneta. Per ogni lancio della moneta il programma deve stampare `Heads` (testa) o `Tails` (croce). Fate lanciare al programma la moneta 100 volte e contate il numero di volte in cui compare ogni lato della moneta. Stampate i risultati. Il programma deve chiamare una funzione separata `flip` che non riceve alcun argomento e restituisce 0 per croce e 1 per testa. Se il programma simula in maniera realistica il lancio della moneta, allora ogni lato della moneta deve comparire approssimativamente la metà delle volte, per un totale di approssimativamente 50 teste e 50 croci.

**5.32 (*Indovina il numero*)** Scrivete un programma in C che realizzi il gioco “indovina il numero” come segue: il vostro programma sceglie il numero da indovinare selezionando a caso un intero nell’intervallo da 1 a 1.000. Il programma quindi stampa:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
```

Il giocatore allora scrive una prima risposta. Il programma risponde con una delle seguenti frasi:

1. Excellent! You guessed the number!  
Would you like to play again (y or n)?
2. Too low. Try again.
3. Too high. Try again.

Se la risposta del giocatore è sbagliata, il vostro programma deve entrare in un ciclo finché il giocatore non indovina finalmente il numero giusto. Deve continuare a dire al giocatore *Too high* o *Too low* per aiutarlo a "convergere" sulla risposta corretta.

**5.33 (*Modifiche a "Indovina il numero"*)** Modificate il programma dell'Esercizio 5.32 per contare il numero delle risposte date dal giocatore. Se il numero è 10 o di meno, stampate "Either you know the secret or you got lucky!". Se il giocatore indovina il numero dopo dieci tentativi, allora stampate "Aha! You know the secret!". Se il giocatore dà più di dieci risposte, allora stampate "You should be able to do better!". Perché dovrebbero volerci non più di 10 risposte? Ebbene, con ogni "risposta buona" il giocatore dovrebbe essere in grado di eliminare la metà dei numeri. Ora mostrate perché un numero da 1 a 1.000 può essere indovinato in 10 o anche in meno tentativi.

**5.34 (*Espozenziazione ricorsiva*)** Scrivete una funzione ricorsiva `power(base, exponent)` che quando viene invocata restituisca

$$\text{base}^{\text{exponent}}$$

Per esempio, `power(3, 4) = 3 * 3 * 3 * 3`. Supponete che `exponent` sia un intero maggiore o uguale a 1. Suggerimento: il passo di ricorsione dovrebbe usare la relazione

$$\text{base}^{\text{exponent}} = \text{base} * \text{base}^{\text{exponent}-1}$$

e la condizione di terminazione si verifica quando `exponent` è uguale a 1 perché

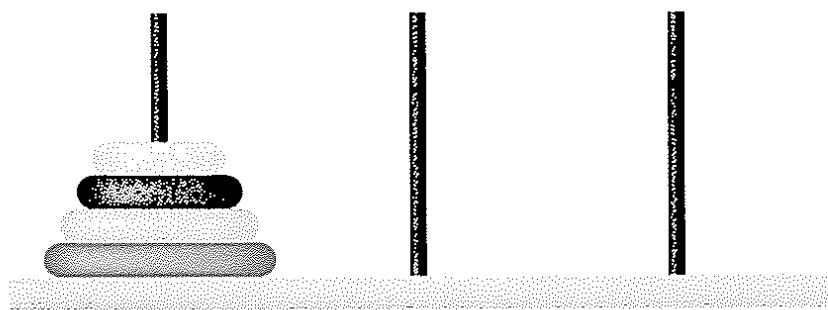
$$\text{base}^1 = \text{base}$$

**5.35 (*Fibonacci*)** La serie di Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

inizia con i termini 0 e 1 e ha la proprietà che ogni termine che segue è la somma dei due termini precedenti. Innanzitutto, scrivete una funzione `fibonacci(n)` non ricorsiva che calcoli l' $n^{\text{mo}}$  numero di Fibonacci. Usate `int` per il parametro della funzione e `unsigned long long int` per il suo tipo di ritorno. Poi, determinate il numero di Fibonacci più grande che può essere stampato sul vostro sistema.

**5.36 (*Le Torri di Hanoi*)** Ogni allievo informatico deve prima o poi trovarsi alle prese con certi problemi classici, e quello delle Torri di Hanoi (mostrato nel seguente diagramma) è uno dei più famosi:



La leggenda narra che in un tempio dell'Estremo Oriente alcuni sacerdoti tentano di spostare una pila di dischi da un piolo a un altro. La pila iniziale aveva 64 dischi infilati in un piolo, disposti in dimensione decrescente dal basso verso l'alto. I sacerdoti tentano di spostare la pila da questo piolo a un secondo piolo, con il vincolo che si sposti esattamente un disco alla volta e che nessun disco più grande possa essere collocato sopra un disco più piccolo. Un terzo piolo è disponibile per contenere temporaneamente i dischi. Presumibilmente il mondo finirà quando i sacerdoti porteranno a termine il loro compito, così siamo poco incentivati a facilitare i loro sforzi.

Supponiamo che i sacerdoti tentino di spostare i dischi dal piolo 1 al piolo 3. Vogliamo sviluppare un algoritmo che stampi la sequenza precisa del trasferimento di ogni disco da un piolo all'altro.

Se affrontassimo questo problema coi metodi convenzionali, ci troveremmo subito disperatamente in difficoltà in merito alla disposizione dei dischi. Invece, se affrontiamo il problema con in mente la ricorsione,

questo diventa immediatamente trattabile. Spostare  $n$  dischi si può vedere in termini dello spostamento di solo  $n - 1$  dischi (e da qui la ricorsione) come segue.

- Spostare  $n - 1$  dischi dal piolo 1 al piolo 2, usando il piolo 3 come supporto temporaneo.
- Spostare l'ultimo disco (il più grande) dal piolo 1 al piolo 3.
- Spostare gli  $n - 1$  dischi dal piolo 2 al piolo 3, usando il piolo 1 come supporto temporaneo.

Il processo termina quando l'ultimo compito implica lo spostamento di  $n = 1$  dischi, cioè il caso di base. Ciò si attua spostando banalmente il disco senza la necessità di un supporto temporaneo.

Scrivete un programma per risolvere il problema delle Torri di Hanoi. Usate una funzione ricorsiva con quattro parametri:

- il numero di dischi da spostare;
- il piolo su cui questi dischi sono inizialmente infilati;
- il piolo nel quale spostare questa pila di dischi;
- il piolo da usare come supporto temporaneo.

Il vostro programma deve stampare le istruzioni esatte, necessarie a spostare i dischi dal piolo di partenza al piolo di arrivo. Per esempio, per spostare una pila di tre dischi dal piolo 1 al piolo 3, il vostro programma deve stampare la seguente serie di mosse:

```
1 → 3 (Questo significa spostare un disco dal piolo 1 al piolo 3.)
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3
```

**5.37 (Le Torri di Hanoi: soluzione iterativa)** Qualsiasi programma che si può implementare ricorsivamente si può implementare iterativamente, benché talvolta con difficoltà considerevolmente maggiore e con chiarezza considerevolmente minore. Cercate di scrivere una versione iterativa delle Torri di Hanoi. Se ci riuscite, confrontate la vostra versione iterativa con la versione ricorsiva che avete sviluppato nell'Esercizio 5.36. Individuate i problemi di prestazioni e di chiarezza e verificate la vostra abilità nel dimostrare la correttezza dei programmi.

**5.38 (Visualizzare la ricorsione)** È interessante osservare la ricorsione “in azione”. Modificate la funzione fattoriale della Figura 5.9 per stampare la sua variabile locale e il parametro della chiamata ricorsiva. Per ogni chiamata ricorsiva stampate gli output su una riga separata e aggiungete un livello di indentazione. Fate del vostro meglio per rendere gli output chiari, interessanti e significativi. L'obiettivo qui è quello di progettare e implementare un formato di output che aiuti una persona a capire meglio la ricorsione. Se volete, potete aggiungere tali capacità di stampa ai molti altri esempi ed esercizi di ricorsione presenti in tutto il testo.

**5.39 (Massimo comun divisore ricorsivo)** Il massimo comun divisore degli interi  $x$  e  $y$  è l'intero più grande che divide in parti uguali sia  $x$  che  $y$ . Scrivete una funzione ricorsiva gcd che restituiscia il massimo comun divisore di  $x$  e  $y$ . Il massimo comun divisore di  $x$  e  $y$  è definito ricorsivamente come segue: se  $y$  è uguale a 0, allora  $\text{gcd}(x, y)$  è  $x$ ; altrimenti  $\text{gcd}(x, y)$  è  $\text{gcd}(y, x \% y)$ , dove  $\%$  è l'operatore di resto.

**5.40 (main ricorsiva)** Si può chiamare la funzione main ricorsivamente? Scrivete un programma contenente una funzione main. Includete la variabile locale static count inizializzata a 1. Postincrementate e stampate il valore di count ogni volta che main è chiamata. Fate eseguire il programma. Che cosa accade?

**5.41 (Distanza tra punti)** Scrivete una funzione distance che calcoli la distanza tra due punti  $(x_1, y_1)$  e  $(x_2, y_2)$ . Tutti i numeri e i valori di ritorno devono essere di tipo double.

**5.42** Che cosa fa il seguente programma? Che cosa accade se scambiate le righe 7 e 8?

```
1 #include <stdio.h>
2
3 int main(void) {
4 int c = '\0'; // variabile per tenere il carattere inserito dall'utente
```

```

5 if ((c = getchar()) != EOF) {
6 main();
7 printf("%c", c);
8 }
10 }
```

5.43 Che cosa fa il seguente programma?

```

1 #include <stdio.h>
2
3 int mystery(int a, int b); // prototipo di funzione
4
5 int main(void) {
6 printf("%s", "Enter two positive integers: ");
7 int x = 0; // primo intero
8 int y = 0; // secondo intero
9 scanf("%d%d", &x, &y);
10
11 printf("The result is %d\n", mystery(x, y));
12 }
13
14 // Il parametro b deve essere un intero positivo
15 // per prevenire la ricorsione infinita
16 int mystery(int a, int b) {
17 // caso di base
18 if (1 == b) {
19 return a;
20 }
21 else { // passo ricorsivo
22 return a + mystery(a, b - 1);
23 }
24 }
```

5.44 Dopo aver determinato che cosa fa il programma dell'Esercizio 5.43, modificate il programma perché funzioni correttamente dopo aver rimosso la restrizione che prevede la positività del secondo argomento.

5.45 (*Verificare le funzioni della libreria math*) Scrivete un programma che verifichi le funzioni della libreria math mostrate nella tabella del Paragrafo 5.3. Provate ognuna di queste funzioni con il vostro programma, stampando tabelle dei valori di ritorno per diversi valori degli argomenti.

5.46 Trovate l'errore in ognuno dei seguenti segmenti di programma e spiegate come correggerlo.

- `double cube(float); // prototipo di funzione  
cube(float number) { // definizione di funzione  
 return number * number * number;  
}`
- `int randomNumber = srand();`
- `double y = 123.45678;  
int x;  
x = y;  
printf("%f\n", (double) x);`
- `double square(double number) {  
 double number;  
 return number * number;  
}`

```
e) int sum(int n) {
 if (0 == n) {
 return 0;
 }
 else {
 return n + sum(n);
 }
}
```

**5.47 (Modifiche al gioco Craps)** Modificate il programma del gioco Craps della Figura 5.7 per consentire di scommettere. Impacchettate come funzione la porzione del programma che esegue un giro del gioco Craps. Inizializzate la variabile bankBalance a 1.000 dollari. Richiedete al giocatore di inserire una scommessa come valore per la variabile wager. Usate un ciclo while per controllare che wager sia minore o uguale a bankBalance e, se non lo è, richiedete all'utente di reinserire il valore di wager finché non viene inserito un valore valido. Dopo l'inserimento di un valore valido di wager, fate eseguire un giro di Craps. Se il giocatore vince, aumentate il valore di bankBalance della quantità pari al valore di wager e stampate il nuovo bankBalance. Se il giocatore perde, diminuite il valore di bankBalance della quantità pari al valore wager, stampate il nuovo bankBalance, controllate se bankBalance è diventato zero e, se è così, stampate il messaggio "Sorry. You busted!". Man mano che il gioco prosegue, stampate vari messaggi per creare un certo "dialogo", come "Oh, you're going for broke, huh?" o "Aw c'mon, take a chance!" o "You're up big. Now's the time to cash in your chips!".

**5.48 (Progetto di ricerca: migliorare l'implementazione ricorsiva della funzione di Fibonacci)** Nel Paragrafo 5.15, l'algoritmo ricorsivo usato per calcolare i numeri di Fibonacci era intuitivamente attraente. Tuttavia, ricordate che l'algoritmo produce un'esplosione esponenziale delle chiamate della funzione ricorsiva. Ricercate online l'implementazione ricorsiva della funzione di Fibonacci. Studiate i vari approcci, compresa la versione iterativa nell'Esercizio 5.35 e le versioni che usano soltanto la cosiddetta "ricorsione di coda". Esaminatene i relativi meriti.

### Istruzione assistita da computer

I computer creano possibilità entusiasmanti per migliorare l'esperienza formativa degli studenti di tutto il mondo, come suggerito dai prossimi cinque esercizi. [Nota: controllate iniziative come il progetto "One Laptop Per Child" ([www.laptop.org](http://www.laptop.org)).]

**5.49 (Istruzione assistita da computer)** L'uso di computer nell'ambito dell'istruzione è chiamato *istruzione assistita da computer* (CAI, *computer-assisted instruction*). Scrivete un programma che aiuti uno studente di scuola elementare a imparare la moltiplicazione. Usate la funzione rand per generare due interi positivi di una cifra. Il programma deve presentare all'utente una domanda del tipo

How much is 6 times 7?

Lo studente inserisce quindi la risposta e il programma la controlla. Se è corretta, stampate il messaggio "Very good!" e ponete come domanda un'altra moltiplicazione. Se la risposta è sbagliata, stampate il messaggio "No. Please try again." e lasciate che lo studente tenti di rispondere alla stessa domanda ripetutamente, finché non risponde in modo esatto. Si deve usare una funzione separata per generare ogni nuova domanda. Questa funzione deve essere chiamata una volta quando l'applicazione inizia l'esecuzione e ogni volta che l'utente risponde correttamente alla domanda.

**5.50 (Istruzione assistita da computer: ridurre l'affaticamento dello studente)** Un problema degli ambienti CAI riguarda l'affaticamento dello studente. Questo può essere ridotto variando le risposte del computer per tenere desta l'attenzione. Modificate il programma dell'Esercizio 5.49 così che siano stampati vari commenti per ogni risposta, come segue.

Possibili commenti per una risposta corretta:

```
Very good!
Excellent!
Nice work!
Keep up the good work!
```

Possibili commenti per una risposta sbagliata:

No. Please try again.  
Wrong. Try once more.  
Don't give up!  
No. Keep trying.

Utilizzate la generazione di numeri casuali per scegliere un numero da 1 a 4 da usare per selezionare uno dei quattro commenti appropriati per ogni risposta corretta o sbagliata. Usate un'istruzione `switch` per selezionare le risposte.

**5.51 (Istruzione assistita da computer: monitorare le prestazioni dello studente)** I sistemi più sofisticati di istruzione assistita da computer monitorano le prestazioni dello studente per un periodo di tempo. La decisione di cominciare un nuovo argomento si basa spesso sul successo dello studente negli argomenti precedenti. Modificate il programma dell'Esercizio 5.50 per contare il numero delle risposte esatte e sbagliate scritte dallo studente. Dopo che lo studente scrive 10 risposte, il vostro programma deve calcolare la percentuale di quelle corrette. Se la percentuale è inferiore al 75%, stampate "Please ask your teacher for extra help.", quindi reiniziate il programma così che possa provarci un altro studente. Se la percentuale è del 75% o maggiore, stampate "Congratulations, you are ready to go to the next level!", quindi reiniziate il programma così che possa provarci un altro studente.

**5.52 (Istruzione assistita da computer: livelli di difficoltà)** Gli Esercizi 5.49–5.51 richiedono di sviluppare un programma di istruzione assistita da computer per aiutare a insegnare la moltiplicazione a uno studente di scuola elementare. Modificate il programma per permettere all'utente di inserire un livello di difficoltà: a un livello di difficoltà 1, il programma deve usare nei problemi solo numeri a cifra singola; a un livello di difficoltà 2, numeri a due cifre, e così via.

**5.53 (Istruzione assistita da computer: variare i tipi di problemi)** Modificate il programma dell'Esercizio 5.52 per consentire all'utente di selezionare un tipo di problema aritmetico da studiare. Un'opzione 1 significa soltanto problemi di addizione, 2 significa soltanto problemi di sottrazione, 3 significa soltanto problemi di moltiplicazione e 4 significa un mix casuale di tutti questi tipi.

## Caso pratico di simulazione di numeri casuali: la gara tra la tartaruga e la lepre

**5.54 (La gara tra la tartaruga e la lepre)** In questo esercizio creerete la classica gara tra la lepre e la tartaruga. Utilizzeremo la generazione casuale dei numeri per sviluppare una simulazione di questo evento memorabile.

I nostri concorrenti iniziano la gara al primo di 70 quadrati. Ogni quadrato rappresenta una posizione nel percorso della gara. La linea di arrivo è al quadrato 70. Il primo concorrente che raggiunge o sorpassa il quadrato 70 vincerà un cesto di carote e lattuga. Dato che la corsa si svolge in montagna, a volte i concorrenti scivoleranno perdendo così terreno.

Un orologio scandisce i secondi. A ogni secondo dell'orologio, la vostra applicazione deve correggere la posizione degli animali in base alle seguenti regole.

| Animale   | Tipo di mossa     | Percentuale di tempo | Mossa reale          |
|-----------|-------------------|----------------------|----------------------|
| Tartaruga | Passo veloce      | 50%                  | 3 quadrati avanti    |
|           | Scivolata         | 20%                  | 6 quadrati indietro  |
|           | Passo lento       | 30%                  | 1 quadrato avanti    |
| Lepre     | Dorme             | 20%                  | Nessuno spostamento  |
|           | Grande salto      | 20%                  | 9 quadrati avanti    |
|           | Grande scivolata  | 10%                  | 12 quadrati indietro |
|           | Piccolo salto     | 30%                  | 1 quadrato avanti    |
|           | Piccola scivolata | 20%                  | 2 quadrati indietro  |

Fate uso di variabili per tenere traccia delle posizioni degli animali (numeri da 1 a 70). Ogni animale inizia la gara alla posizione 1. Se uno degli animali scivola all'indietro oltre la posizione 1, rimettetelo in posizione 1. Se un animale supera il quadrato 70, riportatelo al quadrato 70.

Generate le percentuali elencate nella precedente tabella producendo un intero casuale,  $x$ , nell'intervallo  $1 \leq x \leq 10$ . La tartaruga effettuerà un "passo veloce" quando  $1 \leq x \leq 5$ , una "scivolata" quando  $6 \leq x \leq 7$  e un "passo lento" quando  $8 \leq x \leq 10$ . Usate uno schema simile per la lepre.

Iniziate la gara visualizzando

ON YOUR MARK, GET SET  
BANG !!!!  
AND THEY'RE OFF !!!!

Dopodiché, a ogni secondo che passa (ossia, a ogni iterazione di un ciclo) visualizzate una riga di 70 posizioni, mostrando la lettera T nella posizione della tartaruga e la lettera H nella posizione della lepre. Se i concorrenti finiscono nello stesso quadrato, la tartaruga morde la lepre e il programma deve visualizzare "OUCH!!!". In quella posizione. Tutte le posizioni al di fuori della T, della H o di OUCH!!! (in caso di parità) devono rimanere vuote.

Ogni volta che visualizzate una riga, controllate se uno degli animali ha raggiunto o oltrepassato il quadrato 70. In questo caso, visualizzate il vincitore e terminate la simulazione. Se vince la tartaruga, stampate "TORTOISE WINS!!! YAY!!!". Se vince la lepre, stampate "Hare wins. Yuch". Se entrambi gli animali vincono nello stesso momento, potete favorire la tartaruga (quella svantaggiata), o potete visualizzare "It's a tie". Se nessuno dei due animali vince, eseguite di nuovo il ciclo per simulare il prossimo ticchettio dell'orologio. Quando siete pronti a eseguire la vostra applicazione, radunate un gruppo di amici per vedere la gara. Rimarrete sorpresi dal coinvolgimento degli spettatori!

## CAPITOLO

# 6

# Array

### Sommario del capitolo

- 6.1 Introduzione
- 6.2 Array
- 6.3 Definire gli array
- 6.4 Esempi di array
- 6.5 Uso di array di caratteri per memorizzare e manipolare stringhe
- 6.6 Array locali statici e array locali automatici
- 6.7 Passare gli array alle funzioni
- 6.8 Ordinamento di array
- 6.9 Caso pratico di introduzione alla data science: analisi dei dati di un sondaggio
- 6.10 Ricerca in array
- 6.11 Array multidimensionali
- 6.12 Array di lunghezza variabile
- 6.13 Programmazione sicura in C
- 6.14 Riepilogo

### Obiettivi

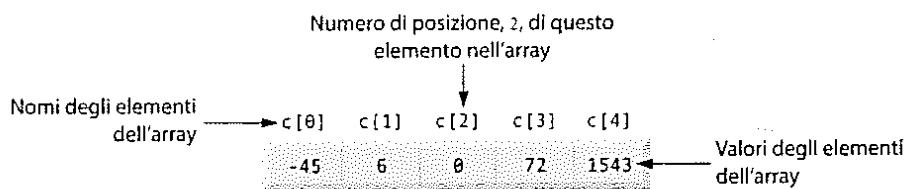
- Usare la struttura di dati array per rappresentare liste e tabelle di valori.
- Definire un array, inizializzarlo e fare riferimento ai singoli elementi che lo compongono.
- Definire costanti simboliche.
- Passare gli array alle funzioni.
- Usare gli array per memorizzare, ordinare ed effettuare ricerche in liste e tabelle di valori.
- Introdurre alla data science usando le statistiche descrittive di base, come media, mediana e moda.
- Definire e manipolare array multidimensionali.
- Creare array di lunghezza variabile la cui dimensione è determinata al momento dell'esecuzione.
- Comprendere i problemi relativi alla sicurezza legati all'input con `scanf`, all'output con `printf` e agli array.

### 6.1 Introduzione

Questo capitolo serve da introduzione alle strutture di dati. Gli **array** sono strutture di dati costituite da dati correlati e dello stesso tipo. Nel Capitolo 10 esamineremo il costrutto `struct` del C, una struttura di dati costituita da dati correlati di tipi possibilmente differenti. Array e `struct` sono entità “statiche” in quanto rimangono della stessa dimensione per tutto il loro tempo di vita.

## 6.2 Array

Un array è un gruppo di **elementi** dello stesso tipo immagazzinati in modo contiguo nella memoria. Il diagramma seguente mostra un array di interi chiamato **c**, contenente cinque elementi:



Per fare riferimento a una particolare locazione o elemento nell'array, si specifica il nome dell'array, seguito dal **numero di posizione** dell'elemento racchiuso tra parentesi quadre ([ ]). La posizione del primo elemento corrisponde alla posizione numero 0 (zero). Il numero di posizione è chiamato **indice** dell'elemento. Un indice deve essere un intero non negativo oppure un'espressione intera.

Analizziamo più da vicino l'array nel diagramma precedente. Il **nome** dell'array è **c**. Il **valore** di **c[0]** è -45, di **c[2]** è 0 e di **c[4]** è 1543. Un nome di array con indice è un *lvalue* che può essere usato sul lato sinistro di un'assegnazione. Pertanto, l'istruzione:

```
c[2] = 1000;
```

sostituisce il valore corrente di **c[2]**, ossia (0), con il valore 1000. Per stampare la somma dei valori contenuti nei primi tre elementi dell'array **c** scriviamo:

```
printf("%d", c[0] + c[1] + c[2]);
```

Per dividere il valore dell'elemento 3 dell'array **c** per 2 e assegnare il risultato alla variabile **x** scriviamo:

```
x = c[3] / 2;
```

Le parentesi usate per racchiudere l'indice di un array sono un operatore con il livello più alto di precedenza. La tabella seguente mostra la precedenza e l'associatività degli operatori introdotti fino a questo punto nel testo.

| Operatori                                | Associatività        | Tipo                |
|------------------------------------------|----------------------|---------------------|
| [ ] () ++ (postfisso) -- (postfisso)     | da sinistra a destra | precedenza più alta |
| + - ! ++ (prefisso) -- (prefisso) (tipo) | da destra a sinistra | unario              |
| * / %                                    | da sinistra a destra | moltiplicativo      |
| +                                        | da sinistra a destra | additivo            |
| < <= > >=                                | da sinistra a destra | relazionale         |
| == !=                                    | da sinistra a destra | di uguaglianza      |
| &&                                       | da sinistra a destra | AND logico          |
|                                          | da sinistra a destra | OR logico           |
| ? :                                      | da destra a sinistra | condizionale        |
| = += -= *= /= %=                         | da destra a sinistra | di assegnazione     |
| ,                                        | da sinistra a destra | virgola             |

### ✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- Si può fare riferimento all'elemento di un array fornendo il nome dell'array, seguito dal numero di posizione dell'elemento racchiuso tra parentesi quadre ([]).
- Il primo elemento in ogni array ha numero di posizione 1.
- Il numero di posizione tra le parentesi quadre è chiamato indice dell'elemento, e deve essere un intero o un'espressione intera.

**Risposta:** b) è *falsa*. In realtà, il primo elemento in ogni array ha numero di posizione 0.

2. (*Codice*) Scrivete un'istruzione che stampi il prodotto di tipo `int` dei valori contenuti nei primi quattro elementi dell'array di interi `grades`.

**Risposta:** `printf("%d", grades[0] * grades[1] * grades[2] * grades[3]);`

## 6.3 Definire gli array

Per definire un array, dovete specificare il tipo di elemento e il numero di elementi dell'array in modo che il compilatore possa riservare la giusta quantità di memoria. La definizione seguente riserva cinque elementi per l'array di interi `c`, che ha indici nell'intervallo 0-4.

```
int c[5];
```

Le definizioni

```
int b[100];
int x[27];
```

riservano 100 elementi per l'array intero `b` e 27 elementi per l'array intero `x`. Questi array hanno indici, rispettivamente, negli intervalli 0-99 e 0-26.

Un array di tipo `char` può memorizzare una stringa di caratteri. Le stringhe di caratteri e la loro somiglianza con gli array saranno trattate nel Capitolo 8. La relazione tra puntatori e array sarà esaminata nel Capitolo 7.

### ✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- Quando create un array, dovete specificare il tipo e il numero di elementi dell'array, in modo che il compilatore possa riservare la giusta quantità di memoria.
- La definizione seguente riserva spazio per l'array `temperatures` di tipo `double`, che ha indici nell'intervallo 0-6:

```
double temperatures[7];
```

- La definizione seguente riserva 50 elementi per l'array `b` di tipo `float` e 19 elementi per l'array `x` di tipo `float`:

```
float b[50];
float x[19];
```

- Un array di tipo `string` può memorizzare una stringa di caratteri.

**Risposta:** d) è *falsa*. In realtà, un array di tipo `char` può memorizzare una stringa di caratteri. Non esiste nel C un tipo `string`.

## 6.4 Esempi di array

Questa sezione presenta vari esempi che illustrano come definire e inizializzare gli array e come effettuare molte manipolazioni comuni.

### 6.4.1 Definire un array e usare un ciclo per impostare i valori degli elementi di un array

Come le altre variabili locali, gli elementi di un array non inizializzati contengono valori spazzatura. Il programma della Figura 6.1 usa l'istruzione `for` per impostare gli elementi di un array `n` intero di 5 elementi con valori uguali a zero (righe 10-12) e per stampare l'array in formato tabellare (righe 17-19). La prima istruzione `printf` (riga 14) stampa le intestazioni per le due colonne stampate nella successiva istruzione `for`.

```

1 // fig06_01.c
2 // Inizializzare gli elementi di un array a zero.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7 int n[5]; // n e' un array di 5 interi
8
9 // imposta gli elementi dell'array n a 0
10 for (size_t i = 0; i < 5; ++i) {
11 n[i] = 0; // imposta a 0 l'elemento alla locazione i
12 }
13
14 printf("%s%8s\n", "Element", "Value");
15
16 // stampa il contenuto dell'array n in formato tabellare
17 for (size_t i = 0; i < 5; ++i) {
18 printf("%zu%8d\n", i, n[i]);
19 }
20 }
```

| Element | Value |
|---------|-------|
| 0       | 0     |
| 1       | 0     |
| 2       | 0     |
| 3       | 0     |
| 4       | 0     |

**Figura 6.1** Inizializzare gli elementi di un array a zero.

La variabile di controllo contatore `i` è di tipo `size_t` in ogni istruzione `for` (righe 10 e 17). Secondo il C standard `size_t` rappresenta un tipo intero senza segno ed è raccomandato per qualsiasi variabile che rappresenta la dimensione o gli indici di un array. Il tipo `size_t` è definito nel file di intestazione `<stddef.h>`, ed è spesso incluso in altri file di intestazione (come `<stdio.h>`).<sup>1</sup> Per stampare i valori di tipo `size_t` si usa la specifica di conversione `%zu`.

1. Se durante la compilazione del programma della Figura 6.1 ricevete la segnalazione di errori, includete `<stddef.h>` nel vostro programma.

### 6.4.2 Inizializzare un array in una definizione con una lista di inizializzatori

È possibile inizializzare gli elementi di un array quando l'array viene definito fornendo una lista di **inizializzatori dell'array** separati da virgole racchiusa tra parentesi graffe, {}. Il programma della Figura 6.2 inizializza un array intero con cinque valori (riga 7) e lo stampa in formato tabellare.

```

1 // fig06_02.c
2 // Inizializzare gli elementi di un array con una lista di inizializzatori.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7 int n[5] = {32, 27, 64, 18, 95}; // inizializza n con lista di inizializzatori
8
9 printf("%5%8s\n", "Element", "Value");
10
11 // stampa il contenuto dell'array in formato tabellare
12 for (size_t i = 0; i < 5; ++i) {
13 printf("%7zu%8d\n", i, n[i]);
14 }
15 }
```

| Element | Value |
|---------|-------|
| 0       | 32    |
| 1       | 27    |
| 2       | 64    |
| 3       | 18    |
| 4       | 95    |

Figura 6.2 Inizializzare gli elementi di un array con una lista di inizializzatori.

Se vi sono meno inizializzatori degli elementi nell'array, i restanti elementi sono inizializzati a 0. Per esempio, gli elementi dell'array n nella Figura 6.1 avrebbero potuto essere inizializzati a zero come segue:

```
int n[5] = {0}; // inizializza l'intero array a zero
```

Questo simbolo indica che questa è una buona pratica. Questa definizione inizializza esplicitamente n[0] a 0 e inizializza implicitamente i restanti elementi a 0. Fornire più inizializzatori nella lista di inizializzatori di quanti sono gli elementi nell'array è un errore di compilazione. Per esempio, la seguente definizione di un array genera un errore di compilazione poiché ci sono quattro inizializzatori ma solo tre elementi dell'array:

```
int n[3] = {32, 27, 64, 18};
```

La definizione seguente crea un array di cinque elementi inizializzati con i valori 1-5:

```
int n[] = {1, 2, 3, 4, 5};
```

Se viene omessa la dimensione dell'array, il compilatore calcola il numero degli elementi dell'array sulla base del numero di inizializzatori.

### 6.4.3 Specificare la dimensione di un array con una costante simbolica e inizializzare gli elementi dell'array con espressioni da calcolare

Il programma della Figura 6.3 inizializza gli elementi di un array s di 5 elementi con i valori 2, 4, 6, 8 e 10, poi stampa l'array in formato tabellare. I valori sono generati moltiplicando il contatore del ciclo per 2 e aggiungendo 2.

```

1 // fig06_03.c
2 // Inizializzare gli elementi dell'array s con gli interi pari da 2 a 10.
3 #include <stdio.h>
4 #define SIZE 5 // massima dimensione dell'array
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8 // la costante simbolica SIZE specifica la dimensione dell'array
9 int s[SIZE] = {0}; // l'array s ha un numero di elementi uguale a SIZE
10
11 for (size_t j = 0; j < SIZE; ++j) { // imposta i valori
12 s[j] = 2 + 2 * j;
13 }
14
15 printf("%s%8s\n", "Element", "Value");
16
17 // stampa il contenuto dell'array s in formato tabellare
18 for (size_t j = 0; j < SIZE; ++j) {
19 printf("%zu%8d\n", j, s[j]);
20 }
21 }

```

| Element | Value |
|---------|-------|
| 0       | 2     |
| 1       | 4     |
| 2       | 6     |
| 3       | 8     |
| 4       | 10    |

**Figura 6.3** Inizializzare gli elementi dell'array s con gli interi pari da 2 a 10.

La riga 4 utilizza la **direttiva per il preprocessore #define**.

#define SIZE 5

per creare la **costante simbolica** SIZE con valore 5. Una costante simbolica è un identificatore che è sostituito dal preprocessore del C con un **testo di sostituzione** prima che il programma sia compilato. In questo programma, il preprocessore sostituisce tutte le occorrenze di SIZE con 5.

L'uso di costanti simboliche per specificare le dimensioni di un array rende i programmi più leggibili e modificabili. Nella Figura 6.3, per esempio, potremmo far riempire un array di 1.000 elementi con il primo ciclo for (riga 11) semplicemente cambiando il valore di SIZE nella direttiva #define da 5 a 1000. Senza la costante simbolica, dovremmo modificare il programma nelle righe 9, 11 e 18. Quando i programmi diventano grandi, questa tecnica risulta molto utile per scrivere programmi chiari e di facile lettura e manutenzione; una costante simbolica (come SIZE) è più facile da comprendere del valore numerico 5, che potrebbe avere diversi significati all'interno del codice.

Non si deve terminare una direttiva per il preprocessore #define con un punto e virgola. Se viene fatto questo nella riga 4, il preprocessore sostituisce con il testo "5;" tutte le occorrenze di SIZE. Questo può portare a errori di sintassi in fase di compilazione o a errori logici in fase di esecuzione. Ricordate che il preprocessore non è il compilatore del C.

Assegnare un valore a una costante simbolica in un'istruzione eseguibile è un errore di compilazione; le costanti simboliche non sono variabili. Per convenzione, usate solo lettere maiuscole per i nomi di costanti simboliche, in modo da evidenziarle in un programma e inoltre per ricordarvi che le costanti simboliche non sono variabili.

#### 6.4.4 Sommare gli elementi di un array

Il programma della Figura 6.4 somma i valori contenuti nell'array intero a di cinque elementi. Il corpo dell'istruzione `for` (riga 14) calcola il totale.

```

1 // fig06_04.c
2 // Calcolare la somma degli elementi di un array.
3 #include <stdio.h>
4 #define SIZE 5
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8 // usa una lista di inizializzatori per inizializzare l'array
9 int a[SIZE] = {1, 2, 3, 4, 5};
10 int total = 0; // somma dell'array
11
12 // somma i valori contenuti nell'array a
13 for (size_t i = 0; i < SIZE; ++i) {
14 total += a[i];
15 }
16
17 printf("The total of a's values is %d\n", total);
18 }
```

The total of a's values is 15

**Figura 6.4** Calcolare la somma degli elementi di un array.

#### 6.4.5 Usare array per riassumere i risultati di un sondaggio

Il nostro prossimo esempio usa degli array per riassumere i risultati dei dati raccolti in un sondaggio. Considerate l'enunciato del problema:

*A venti studenti viene chiesto di valutare la qualità del cibo nella caffetteria degli studenti con una scala da 1 a 5 (1 significa pessima e 5 eccellente). Mettete le 20 risposte in un array intero e riassumete i risultati del sondaggio.*

La Figura 6.5 è una tipica applicazione degli array. Vogliamo sommare il numero di ogni tipo di risposte. L'array `responses` (righe 10-11) di 20 elementi contiene le risposte degli studenti. Usiamo un array `frequency` (riga 14) di 6 elementi per contare quante volte ricorre ogni risposta. Ignoriamo `frequency[0]` perché è logico che la risposta 1 incrementi `frequency[1]` anziché `frequency[0]`. Questo ci permette di usare ogni risposta direttamente come indice nell'array `frequency`. Dovete sforzarvi di rendere il programma il più chiaro possibile. A volte può valere la pena sacrificare l'uso efficiente della memoria o del tempo di esecuzione in favore di una scrittura di programmi più chiari. Talvolta le considerazioni relative alle prestazioni prevalgono di gran lunga sulle considerazioni relative alla chiarezza.

```

1 // fig06_05.c
2 // Analisi di un sondaggio di studenti.
3 #include <stdio.h>
4 #define RESPONSES_SIZE 20 // definisci le dimensioni degli array
5 #define FREQUENCY_SIZE 6
6
7 // la funzione main inizia l'esecuzione del programma
8 int main(void) {
9 // inserisci le risposte del sondaggio nell'array delle risposte
10 int responses[RESPONSES_SIZE] =
```

```

11 {1, 2, 5, 4, 3, 5, 2, 1, 3, 1, 4, 3, 3, 3, 2, 3, 3, 2, 2, 5};
12
13 // inizializza i contatori per le frequenze a 0
14 int frequency[FREQUENCY_SIZE] = {0};
15
16 // per ogni risposta, seleziona il valore di un elemento dell'array
17 // responses e usa quel valore come indice nell'array frequency per
18 // determinare l'elemento da incrementare
19 for (size_t answer = 0; answer < RESPONSES_SIZE; ++answer) {
20 ++frequency[responses[answer]];
21 }
22
23 // stampa i risultati
24 printf("%s%12s\n", "Rating", "Frequency");
25
26 // stampa le frequenze in un formato tabellare
27 for (size_t rating = 1; rating < FREQUENCY_SIZE; ++rating) {
28 printf("%zu%12d\n", rating, frequency[rating]);
29 }
30 }
```

| Rating | Frequency |
|--------|-----------|
| 1      | 3         |
| 2      | 5         |
| 3      | 7         |
| 4      | 2         |
| 5      | 3         |

**Figura 6.5** Analisi di un sondaggio di studenti.

### Incrementare i contatori frequency

Il ciclo `for` (righe 19-21) prende le risposte una alla volta dall'array `responses` e incrementa uno dei cinque contatori dell'array `frequency`, da `frequency[1]` a `frequency[5]`. L'istruzione chiave nel ciclo è la riga 20:

```
++frequency[responses[answer]];
```

che incrementa il contatore `frequency` appropriato a seconda del valore dell'espressione `responses[answer]`. Quando la variabile contatore `answer` è 0, `responses[answer]` è 1, quindi `++frequency[responses[answer]]`; è interpretato come

```
++frequency[1];
```

che incrementa `frequency[1]`. Quando `answer` è 1, il valore di `responses[answer]` è 2, quindi `++frequency[responses[answer]]`; è interpretato come

```
++frequency[2];
```

che incrementa `frequency[2]`. Quando `answer` è 2, il valore di `responses[answer]` è 5, quindi `++frequency[responses[answer]]`; è interpretato come

```
++frequency[5];
```

che incrementa `frequency[5]`, e così via.

### Risposte non valide nel sondaggio

A prescindere dal numero di risposte elaborate nel sondaggio, occorre solo un array `frequency` con 6 elementi (ignorando l'elemento zero) per riepilogare i risultati. Ma se i dati contenessero valori non validi, come 13? In questo caso, il programma tenterebbe di aggiungere 1 a `frequency[13]`; questo sarebbe al di fuori dei confini dell'array. Il C non ha meccanismi di controllo dei confini di un array per impedire a un programma di fare riferimento a un elemento che non esiste. Così, un programma che è in esecuzione può "uscire fuori" da una parte o dall'altra di un array senza messaggi di avvertimento (un problema di sicurezza che esamineremo nel Paragrafo 6.13). I programmi devono convalidare la correttezza di tutti i valori in ingresso per evitare che informazioni erronee incidano sui calcoli di un programma.

### Convalidare gli indici degli array

 Fare riferimento a un elemento al di fuori dei limiti di un array è un errore logico. Quando effettuate un ciclo che percorre un array, l'indice dell'array non deve mai andare sotto 0 e deve sempre essere minore del numero totale degli elementi nell'array (dimensione dell'array meno uno). Dovete assicurarvi che tutti i riferimenti all'array rimangano entro i confini dell'array.

## 6.4.6 Rappresentare con grafici a barre i valori degli elementi di un array

Il nostro prossimo esempio (Figura 6.6) legge i numeri da un array e rappresenta le informazioni in un grafico a barre. Mostriamo ciascun numero seguito da una barra formata da altrettanti asterischi. L'istruzione `for` annidata (righe 17-19) stampa le barre iterando `n[i]` volte e stampando un asterisco per ogni iterazione. La riga 21 termina ogni barra.

```

1 // fig06_06.c
2 // Stampa di una grafico a barre.
3 #include <stdio.h>
4 #define SIZE 5
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8 // usa la lista di inizializzatori per inizializzare l'array n
9 int n[SIZE] = {19, 3, 15, 7, 11};
10
11 printf("%s%13s%17s\n", "Element", "Value", "Bar Chart");
12
13 // per ogni elemento dell'array n, stampa una barra del grafico
14 for (size_t i = 0; i < SIZE; ++i) {
15 printf("%zu%13d%8s", i, n[i], "");
16
17 for (int j = 1; j <= n[i]; ++j) { // stampa una barra
18 printf("%c", '*');
19 }
20
21 puts(""); // termina una barra con un newline
22 }
23 }
```

| Element | Value | Bar Chart |
|---------|-------|-----------|
| 0       | 19    | *****     |
| 1       | 3     | ***       |
| 2       | 15    | *****     |
| 3       | 7     | *****     |
| 4       | 11    | *****     |

Figura 6.6 Stampa di un grafico a barre.

#### 6.4.7 Lanciare un dado 60.000.000 di volte e riepilogare i risultati in un array

Nel Capitolo 5 abbiamo detto che avremmo mostrato un metodo più elegante per scrivere il programma del lancio dei dadi della Figura 5.5. Ricordate che il programma ha lanciato un singolo dado a sei facce 60.000.000 di volte e stampato la frequenza risultante per ciascuna faccia del dado. Nella Figura 6.7 è mostrata una versione con array del programma nella Figura 5.5. La riga 17 sostituisce l'intera istruzione `switch` della riga 20 nella Figura 5.5. Di nuovo, usiamo un array `frequency` nel quale ignoriamo l'elemento 0, in modo da poter usare i valori delle facce come indici nell'array.

```

1 // fig06_07.c
2 // Lancia di un dado a sei facce 60.000.000 di volte
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #define SIZE 7
7
8 // la funzione main inizia l'esecuzione del programma
9 int main(void) {
10 srand(time(NULL)); // seme per il generatore di numeri casuali
11
12 int frequency[SIZE] = {0}; // inizializza tutti i conti di frequency a 0
13
14 // lancia il dado 60.000.000 di volte
15 for (int roll = 1; roll <= 60000000; ++roll) {
16 size_t face = 1 + rand() % 6;
17 ++frequency[face]; // sostituisce l'intero switch della Figura 5.5
18 }
19
20 printf("%s%17s\n", "Face", "Frequency");
21
22 // stampa gli elementi di frequency 1-6 in formato tabellare
23 for (size_t face = 1; face < SIZE; ++face) {
24 printf("%4zu%17d\n", face, frequency[face]);
25 }
26 }
```

| Face | Frequency |
|------|-----------|
| 1    | 9997167   |
| 2    | 10003506  |
| 3    | 10001940  |
| 4    | 9995833   |
| 5    | 10000843  |
| 6    | 10000711  |

Figura 6.7 Lancia di un dado a sei facce 60.000.000 di volte.

#### ✓ Autovalutazione

1. (*Codice*) Riscrivete il segmento di codice seguente per definire un array `m` di tipo `double` composto da sette elementi e inizializzate ciascun elemento dell'array a 10:

```

int n[5]; // n e' un array di 5 interi

// imposta gli elementi dell'array n a 0
for (size_t i = 0; i < 5; ++i) {
```

```

 n[i] = 0; // imposta a 0 l'elemento alla locazione i
}

```

Risposta:

```

double m[7]; // m è un array di 7 double

// imposta gli elementi dell'array m a 10.0
for (size_t i = 0; i < 7; ++i) {
 m[i] = 10.0; // imposta a 10.0 l'elemento alla locazione i
}

```

2. (Scelta multipla) Quale delle seguenti affermazioni a), b) o c) è vera?

- a) In un array di interi, se fornite meno inizializzatori rispetto agli elementi dell'array, i restanti elementi sono inizializzati a 0.
- b) È un errore di sintassi fornire una lista con più inizializzatori per un array di quanti siano gli elementi dell'array. Per esempio, `int n[3] = {32, 27, 64, 18};` è un errore di sintassi, poiché ci sono quattro inizializzatori ma solo tre elementi dell'array.
- c) Se la dimensione dell'array è omessa in una definizione con una lista di inizializzatori, il compilatore determina il numero di elementi dell'array in base al numero di elementi nella lista. Quindi, la definizione seguente crea l'array intero s di tre elementi:

```
int s[] = {10, 20, 30};
```

- d) Tutte le affermazioni precedenti sono vere.

Risposta: d.

## 6.5 Uso di array di caratteri per memorizzare e manipolare stringhe

Gli array possono contenere dati di ogni tipo, ma tutti gli elementi di un dato array devono essere dello stesso tipo. Esaminiamo ora la memorizzazione di stringhe in array di caratteri. Finora l'unica funzionalità di elaborazione di stringhe che abbiamo visto è quella relativa alla stampa di una stringa con `printf`. Una stringa come "hello" è in realtà un array di caratteri individuali, più un'altra cosa.

### 6.5.1 Inizializzazione di un array di caratteri con una stringa

Gli array di caratteri hanno diverse caratteristiche particolari. Un array di caratteri può essere inizializzato usando una stringa letterale. Per esempio,

```
char string1[] = "first";
```

Inizializza gli elementi dell'array `string1` con i singoli caratteri nel letterale stringa "first". In questo caso, la dimensione dell'array `string1` è determinata dal compilatore in base alla lunghezza della stringa. La stringa "first" contiene cinque caratteri più un *carattere nullo di terminazione di stringa*. Quindi, `string1` contiene in realtà sei elementi. La sequenza di escape che rappresenta il carattere nullo è '\0'. Tutte le stringhe terminano con questo carattere. Un array di caratteri che rappresenta una stringa deve essere sempre definito di dimensioni abbastanza grandi da contenere il numero di caratteri nella stringa e il carattere nullo che la termina.

### 6.5.2 Inizializzazione di un array di caratteri con una lista di inizializzatori di caratteri

Gli array di caratteri possono anche essere inizializzati con costanti di tipo carattere individuali poste in una lista di inizializzatori, ma ciò può risultare pesante. La definizione precedente è equivalente a

```
char string1[] = {'f', 'i', 'r', 's', 't', '\0'};
```

### 6.5.3 Accesso ai caratteri di una stringa

È possibile accedere ai singoli caratteri di una stringa direttamente usando la notazione con indice per gli array. Quindi, `string1[0]` è il carattere 'f', `string1[3]` è 's' e `string1[5]` è '\0'.

### 6.5.4 Inserimento in un array di caratteri

La definizione seguente crea un array di caratteri che può memorizzare una stringa con *al massimo 19 caratteri e un carattere nullo di terminazione*:

```
char string2[20];
```

L'istruzione

```
scanf("%19s", string2);
```

legge una stringa dalla tastiera e la memorizza in `string2`. Il nome dell'array è passato a `scanf` senza il carattere & usato con variabili che non sono stringhe. Il carattere & si usa normalmente per fornire a `scanf` il riferimento alla *locuzione* di memoria della variabile, così che vi si possa memorizzare un valore. Nel Paragrafo 6.7, quando discuteremo il passaggio di array a funzioni, vedremo anche perché il carattere & non è necessario per i nomi di array.

È vostra responsabilità assicurarvi che l'array nel quale la stringa letta viene memorizzata possa contenere qualunque stringa che l'utente scrive alla tastiera. La funzione `scanf` *non* controlla quanto è grande l'array; legge i caratteri finché non si incontra uno *spazio*, una *tabulazione*, un *newline* o un *indicatore di end-of-file*. La stringa `string2` non deve essere più lunga di 19 caratteri così da lasciare spazio al carattere nullo che la termina. Se l'utente scrive 20 o più caratteri, il programma può arrestarsi o creare un problema di vulnerabilità relativamente alla sicurezza noto come *overflow del buffer*. Per questa ragione, abbiamo usato la specifica di conversione `%19s` in modo che `scanf` legga un massimo di 19 caratteri e non scriva caratteri in memoria oltre la fine dell'array `string2`. (Nel Paragrafo 6.13, rivediamo il potenziale problema di sicurezza causato dall'inserimento in un array di caratteri e analizziamo la funzione `scanf_s`.)



### 6.5.5 Stampa di un array di caratteri che rappresenta una stringa

Un array di caratteri che rappresenta una stringa può essere stampato con `printf` usando la specifica di conversione `%s`. Per esempio, potete stampare l'array di caratteri `string2` con l'istruzione

```
printf("%s\n", string2);
```

Come `scanf`, `printf` non controlla quanto è grande l'array di caratteri e stampa i caratteri della stringa finché non si incontra un carattere nullo di terminazione. [Considerate cosa si stamperebbe se, per qualche ragione, mancasse il carattere nullo di terminazione.]

### 6.5.6 Illustrazione di un array di caratteri

Il programma della Figura 6.8 illustra l'inizializzazione di un array di caratteri con una stringa letterale, la memorizzazione di una stringa letta in un array di caratteri, la stampa di un array di caratteri come una stringa e l'accesso ai caratteri individuali di una stringa. Il programma usa l'istruzione `for` (righe 20-22) per percorrere l'array `string1` e stampare i caratteri individuali separati da spazi, usando la specifica di conversione `%c`. La condizione nell'istruzione `for` è vera finché il contatore è minore della dimensione dell'array e finché non si incontra il carattere nullo di terminazione nella stringa. In questo programma leggiamo solo stringhe che non contengono caratteri di spaziatura. Mostreremo come leggere stringhe con tali caratteri nel Capitolo 8.

```
1 // fig06_08.c
2 // Trattare gli array di caratteri come stringhe.
3 #include <stdio.h>
4 #define SIZE 20
5
6 // la funzione main inizia l'esecuzione del programma
```

```

7 int main(void) {
8 char string1[SIZE] = ""; // riserva 20 caratteri
9 char string2[] = "string literal"; // riserva 15 caratteri
10
11 // memorizza la stringa inserita dall'utente nell'array string1
12 printf("%s", "Enter a string (no longer than 19 characters): ");
13 scanf("%19s", string1); // leggi non più di 19 caratteri
14
15 // stampa le stringhe
16 printf("string1 is: %s\nstring2 is: %s\n", string1, string2);
17 puts("string1 with spaces between characters is:");
18
19 // stampa i caratteri finché non si raggiunge il carattere nullo
20 for (size_t i = 0; i < SIZE && string1[i] != '\0'; ++i) {
21 printf("%c ", string1[i]);
22 }
23
24 puts("");
25 }
```

```

Enter a string (no longer than 19 characters): Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

**Figura 6.8** Trattare gli array di caratteri come stringhe.

### ✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è *falsa*?
  - a) Un array di caratteri che rappresenta una stringa può essere stampato con `printf` e lo specificatore di conversione `%s`, come in:

```
printf("%s\n", month);
```

  - b) La funzione `printf`, come `scanf`, non controlla quanto è grande l'array di caratteri.
  - c) Quando la funzione `printf` stampa i caratteri di un array di caratteri che rappresenta una stringa, termina quando cerca di stampare il primo carattere oltre la fine dell'array.
  - d) Tutte le affermazioni precedenti sono *vere*.

**Risposta:** c) è *falsa*. In realtà, la funzione `printf` continua a stampare i caratteri finché non incontra un carattere nullo di terminazione, anche se è ben oltre la fine dell'array.

2. (*Vero/Falso*) L'array seguente può memorizzare una stringa con al massimo 20 caratteri e un carattere nullo di terminazione:

```
char name1[20];
```

**Risposta:** *Falso*. L'istruzione crea un array di caratteri che può memorizzare una stringa con al massimo 19 caratteri e un carattere nullo di terminazione.

## 6.6 Array locali statici e array locali automatici

Il Capitolo 5 ha esaminato lo specificatore della classe di memoria `static`. Una variabile locale `static` esiste per la durata del programma, ma è visibile soltanto nel corpo di una funzione. Possiamo applicare `static` alla definizione di un array locale in modo che l'array non debba essere creato e inizializzato ogni volta che la funzione è chiamata e che non venga distrutto ogni volta che si esce dalla funzione. Questo riduce il tempo di esecuzione del programma, soprattutto per i programmi con funzioni chiamate frequentemente e che contengono array grandi. Gli array che sono `static` sono inizializzati una volta per tutte all'avvio del programma. Se non inizializzate esplicitamente un array `static`, gli elementi di quell'array vengono automaticamente inizializzati a zero.

 Il programma della Figura 6.9 illustra la funzione `staticArrayInit` (righe 21-38) con un array locale `static` (riga 23) e la funzione `automaticArrayInit` (righe 41-58) con un array locale automatico (riga 43). La funzione `staticArrayInit` è chiamata due volte (righe 11 e 15). L'array locale `static` nella funzione è inizializzato a zero prima dell'avvio del programma (riga 23). La funzione stampa l'array, aggiunge 5 a ogni elemento e stampa di nuovo l'array. La seconda volta che la funzione è chiamata, l'array locale `static` contiene i valori che sono stati memorizzati nel corso della prima chiamata.

```

1 // fig06_09.c
2 // Gli array statici sono automaticamente inizializzati a zero.
3 #include <stdio.h>
4
5 void staticArrayInit(void); // prototipo di funzione
6 void automaticArrayInit(void); // prototipo di funzione
7
8 // la funzione main inizia l'esecuzione del programma
9 int main(void) {
10 puts("First call to each function:");
11 staticArrayInit();
12 automaticArrayInit();
13
14 puts("\n\nSecond call to each function:");
15 staticArrayInit();
16 automaticArrayInit();
17 puts("");
18 }
19
20 // funzione usata per illustrare un array locale statico
21 void staticArrayInit(void) {
22 // inizializza gli elementi a 0 prima che la funzione sia chiamata
23 static int array1[3];
24
25 puts("\nValues on entering staticArrayInit:");
26
27 // stampa i contenuti di array1
28 for (size_t i = 0; i <= 2; ++i) {
29 printf("array1[%zu] = %d ", i, array1[i]);
30 }
31
32 puts("\nValues on exiting staticArrayInit:");
33
34 // modifica e stampa i contenuti di array1
35 for (size_t i = 0; i <= 2; ++i) {
36 printf("array1[%zu] = %d ", i, array1[i] += 5);
37 }

```

```

38 }
39 // funzione usata per illustrare un array locale automatico
40 void automaticArrayInit(void) {
41 // inizializza gli elementi ogni volta che la funzione e' chiamata
42 int array2[3] = {1, 2, 3};
43
44 puts("\n\nValues on entering automaticArrayInit:");
45
46 // stampa i contenuti di array2
47 for (size_t i = 0; i <= 2; ++i) {
48 printf("array2[%zu] = %d ", i, array2[i]);
49 }
50
51 puts("\nValues on exiting automaticArrayInit:");
52
53 // modifica e stampa i contenuti di array2
54 for (size_t i = 0; i <= 2; ++i) {
55 printf("array2[%zu] = %d ", i, array2[i] += 5);
56 }
57 }
58 }
```

First call to each function:

```

Values on entering staticArrayInit:
array1[0] = 0 array1[1] = 0 array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5
```

```

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8
```

Second call to each function:

```

Values on entering staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5 — valori preservati dall'ultima chiamata
Values on exiting staticArrayInit:
array1[0] = 10 array1[1] = 10 array1[2] = 10
```

```

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3 — valori reinizializzati dopo l'ultima chiamata
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8
```

**Figura 6.9** Gli array statici sono automaticamente inizializzati a zero se non vengono esplicitamente inizializzati.

Anche la funzione `automaticArrayInit` è chiamata due volte (righe 12 e 16). Gli elementi dell'array locale automatico sono inizializzati con i valori 1, 2 e 3 (riga 43). La funzione stampa l'array, aggiunge 5 a ogni elemento e stampa di nuovo l'array. La seconda volta che la funzione è chiamata, gli elementi dell'array sono inizializzati di nuovo a 1, 2 e 3, perché l'array ha una permanenza in memoria automatica.

### ✓ Autovalutazione

1. **(Scelta multipla)** Quale delle seguenti affermazioni a), b) o c) è *falsa*?
  - a) Una variabile locale `static` esiste per la durata del programma, ma è visibile solo nel corpo di una funzione.
  - b) Un array locale `static` è creato e inizializzato una volta sola, non a ogni chiamata della funzione. Questo riduce il tempo di esecuzione del programma, particolarmente per i programmi con funzioni chiamate frequentemente che contengono array di grandi dimensioni.
  - c) Se non viene inizializzato esplicitamente un array `static`, i suoi elementi sono inizializzati a zero per default.
  - d) Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

## 6.7 Passare gli array alle funzioni

Per passare come argomento un array a una funzione dovete specificare il nome dell'array senza alcuna parentesi. Per esempio, se l'array `hourlyTemperatures` è stato definito come

```
int hourlyTemperatures[HOURS_IN_A_DAY];
```

la chiamata di funzione

```
modifyArray(hourlyTemperatures, HOURS_IN_A_DAY)
```

passa l'array `hourlyTemperatures` e la sua dimensione alla funzione `modifyArray`.

Ricordate che in C tutti gli argomenti sono passati per valore. Tuttavia, il C passa automaticamente gli array alle funzioni per riferimento; le funzioni chiamate possono modificare i valori degli elementi negli array originari della funzione chiamante. Vedremo nel Capitolo 7 che ciò non è una contraddizione. Il nome di un array ha come valore l'indirizzo in memoria del primo elemento dell'array. Poiché viene passato l'indirizzo di partenza dell'array, la funzione chiamata sa precisamente dove l'array è memorizzato. Di conseguenza, ogni modifica che la funzione chiamata apporta agli elementi dell'array cambia i valori degli elementi dell'array originario nella funzione chiamante.

### Mostrare che il nome di un array è un indirizzo

Il programma della Figura 6.10 fa vedere che "il valore del nome di un array" è realmente l'*indirizzo* del primo elemento dell'array attraverso la stampa di `array`, `&array[0]` e `&array` usando la **specificificazione di conversione %p** per stampare indirizzi. La specificificazione di conversione `%p` normalmente stampa gli indirizzi come numeri esadecimalesimi, ma ciò dipende dal compilatore. I numeri esadecimalesimi (in base 16) consistono nelle cifre da 0 a 9 e nelle lettere da A a F (queste lettere sono gli equivalenti esadecimalesimi dei numeri decimali 10-15). L'Appendice E (disponibile sulla piattaforma MyLab) presenta un'analisi approfondita delle relazioni tra interi binari (in base 2), ottali (in base 8), decimali (in base 10; interi standard) ed esadecimalesimi. L'output mostra che `array`, `&array` e `&array[0]` hanno lo stesso valore. L'output di questo programma è dipendente dal sistema, ma gli indirizzi sono sempre identici per ciascuna esecuzione del programma su un particolare computer.

```

1 // fig06_10.c
2 // Il nome di un array coincide con l'indirizzo del suo primo elemento.
3 #include <stdio.h>
4
5 // la funzione main inizia l'esecuzione del programma
6 int main(void) {
7 char array[5] = ""; // definisci un array di dimensione 5
8
9 printf(" array = %p\n&array[0] = %p\n &array = %p\n",
10 array, &array[0], &array);
11 }
```

```
array = 0031F930
&array[0] = 0031F930
&array = 0031F930
```

**Figura 6.10** Il nome di un array coincide con l'indirizzo del suo primo elemento.

- ☞ Il passaggio degli array per riferimento ha senso per ragioni di prestazioni. Se gli array fossero passati per valore, verrebbe passata una copia di ogni elemento. Nel caso di array grandi passati frequentemente, ciò comporterebbe un consumo significativo di tempo e memoria per copiare gli array. È possibile passare un array per valore (inserendolo in un costrutto `struct`, come spiegheremo nel Capitolo 10).

### Passaggio di singoli elementi di un array

Sebbene gli array interi vengano passati per riferimento, i singoli elementi dell'array sono passati per valore esattamente come le variabili. Singoli pezzi di dati, come i valori `int`, `float` e `char`, sono chiamati **scalari**. Per passare un elemento di un array a una funzione, usate il nome con indice dell'elemento dell'array come argomento nella chiamata della funzione. Nel Capitolo 7 mostreremo come passare gli scalari (cioè le variabili individuali e gli elementi di un array) alle funzioni per riferimento.

### Parametri array

Perché una funzione riceva un array attraverso la chiamata di funzione, la lista dei parametri della funzione deve specificare che si dovrà ricevere un array. L'intestazione di funzione per la funzione `modifyArray` (di cui abbiamo visto la chiamata precedentemente in questa sezione) si potrebbe scrivere come

```
void modifyArray(int b[], size_t size)
```

per indicare che `modifyArray` si aspetta di ricevere un array di interi nel parametro `b` e il numero di elementi dell'array nel parametro `size`. Il numero di elementi dell'array non è richiesto tra le parentesi dell'array. Se è incluso, il compilatore controlla che sia maggiore di zero e poi lo ignora; specificare una dimensione negativa provoca un errore di compilazione. Quando la funzione chiamata userà il nome `b` dell'array si riferirà all'array originario nella funzione chiamante. Quindi, nella chiamata di funzione:

```
modifyArray(hourlyTemperatures, HOURS_IN_A_DAY)
```

il parametro `b` di `modifyArray` rappresenta `hourlyTemperatures` nella funzione chiamante. Nel Capitolo 7 introdurremo altre notazioni per indicare che una funzione riceve un array. Come vedremo, queste notazioni si basano sull'intima relazione tra array e puntatori.

### Differenza tra il passaggio di un intero array e il passaggio di un elemento di un array

Il programma della Figura 6.11 illustra la differenza tra il passaggio di un intero array e il passaggio di un singolo elemento di un array. Il programma dapprima stampa i cinque elementi dell'array di interi `a` (righe 18-20). Successivamente, passiamo l'array `a` e la sua dimensione a `modifyArray` (riga 24), dove ognuno degli elementi di `a` è moltiplicato per 2 (righe 45-47). Poi, le righe 28-30 stampano il contenuto aggiornato di `a`. Come mostra l'output, `modifyArray` ha infatti modificato gli elementi di `a`. Poi la riga 34 stampa il valore di `a[3]` e la riga 36 lo passa alla funzione `modifyElement`. La funzione moltiplica il suo argomento per 2 (riga 53) e stampa il nuovo valore. Quando la riga 39 in `main` stampa ancora `a[3]`, esso *non* è stato modificato perché gli elementi individuali dell'array sono passati per valore.

```
1 // fig06_11.c
2 // Passaggio di array e di singoli elementi di array a funzioni.
3 #include <stdio.h>
4 #define SIZE 5
5
6 //
7 void modifyArray(int b[], size_t size);
```

```
8 void modifyElement(int e);
9
10 // la funzione main inizia l'esecuzione del programma
11 int main(void) {
12 int a[SIZE] = {0, 1, 2, 3, 4}; // inizializza l'array a
13
14 puts("Effects of passing entire array by reference:\n\nThe "
15 "values of the original array are:");
16
17 // stampa l'array originario
18 for (size_t i = 0; i < SIZE; ++i) {
19 printf("%3d", a[i]);
20 }
21
22 puts(""); // stampa un newline
23
24 modifyArray(a, SIZE); // passa l'array "a" a modifyArray per riferimento
25 puts("The values of the modified array are:");
26
27 // stampa l'array modificato
28 for (size_t i = 0; i < SIZE; ++i) {
29 printf("%3d", a[i]);
30 }
31
32 // stampa il valore di a[3]
33 printf("\n\n\nEffects of passing array element "
34 "by value:\n\nThe value of a[3] is %d\n", a[3]);
35
36 modifyElement(a[3]); // passa l'elemento a[3] per valore
37
38 // stampa il valore di a[3]
39 printf("The value of a[3] is %d\n", a[3]);
40 }
41
42 // nella funzione modifyArray, "b" punta all'array originario "a" in memoria
43 void modifyArray(int b[], size_t size) {
44 // moltiplica ogni elemento dell'array per 2
45 for (size_t j = 0; j < size; ++j) {
46 b[j] *= 2; // modifica in realtà l'array originario
47 }
48 }
49
50 // nella funzione modifyElement, "e" e' una copia locale dell'elemento
51 // dell'array a[3] passato da main
52 void modifyElement(int e) {
53 e *= 2; // moltiplica il parametro per 2
54 printf("Value in modifyElement is %d\n", e);
55 }
```

Effects of passing entire array by reference:

The values of the original array are:

0 1 2 3 4

The values of the modified array are:

0 2 4 6 8

Effects of passing array element by value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6

Figura 6.11 Passaggio di array e di singoli elementi di array a funzioni.

### Uso del qualificatore const per impedire alle funzioni di modificare gli elementi di un array

Vi possono essere situazioni nei vostri programmi in cui a una funzione *non* deve essere permesso di modificare gli elementi di un array. Il qualificatore di tipo **const** (per “costante”) del C può impedire che una funzione modifichi un argomento. Quando il parametro di un array è preceduto dal qualificatore **const**, la funzione tratta gli elementi dell’array come costanti. Ogni tentativo di modificare un elemento dell’array nel corpo della funzione produce un errore in fase di compilazione. Questo è un altro esempio del principio del privilegio minimo. A una funzione non si deve dare la possibilità di modificare un array della funzione chiamante, a meno che non sia assolutamente necessario.

La seguente definizione di una funzione chiamata `tryToModifyArray` usa il parametro `const int b[]` (riga 3) per specificare che l’array `b` è costante e non può essere modificato:

```

1 // nella funzione tryToModifyArray, l'array b e' const. per cui non lo si
2 // puo' usare per modificare il suo argomento array nella funzione chiamante
3 void tryToModifyArray(const int b[]) {
4 b[0] /= 2; // errore
5 b[1] /= 2; // errore
6 b[2] /= 2; // errore
7 }
```

Ognuno dei tentativi della funzione di modificare gli elementi dell’array produce un errore di compilazione. Il qualificatore **const** è esaminato in ulteriori contesti nel Capitolo 7.

## ✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

a) Dato il seguente array `hourlyTemperatures`:

```
int hourlyTemperatures[HOURS_IN_A_DAY];
```

la funzione seguente passa l’array `hourlyTemperatures` e la sua dimensione alla funzione `modify-Array`:

```
modifyArray(hourlyTemperatures, HOURS_IN_A_DAY)
```

b) Ricordate che in C tutti gli argomenti sono passati per valore, quindi il C passa automaticamente gli array alle funzioni per valore.

c) Il nome dell’array ha come valore l’indirizzo del primo elemento dell’array.

- d) Poiché viene passato l'indirizzo del primo elemento dell'array, la funzione chiamata sa precisamente dove l'array è memorizzato.

**Risposta:** b) è falsa. In realtà, il C passa automaticamente gli array alle funzioni per riferimento; le funzioni chiamate possono modificare i valori degli elementi negli array originari delle funzioni chiamanti.

2. (*Discussione*) In base al significato del nome della funzione e delle definizioni dei parametri dell'intestazione di funzione seguente, descrivete meglio che potete cos'è probabile che faccia questa funzione:

```
void modifyArray(int b[], size_t size)
```

**Risposta:** La funzione `modifyArray` si aspetta di ricevere un array di interi nel parametro `b` e il numero di elementi dell'array nel parametro `size`. L'array è automaticamente passato per riferimento, di conseguenza la funzione può modificare l'array originario nella funzione chiamante.

## 6.8 Ordinamento di array

Ordinare i dati (cioè mettere i dati in ordine crescente o decrescente) è una delle più importanti applicazioni informatiche. Una banca mette in ordine tutti gli assegni per numero di conto corrente, in modo che, alla fine di ogni mese, è in grado di preparare gli estratti conto bancari individuali. Le compagnie telefoniche ordinano gli elenchi dei clienti per cognome/nome in modo che sia facile trovare i numeri telefonici. Virtualmente, ogni organizzazione deve mettere in ordine dei dati, e in molti casi si tratta di enormi quantità. L'ordinamento di dati è un problema interessante che ha attirato alcuni degli sforzi di ricerca più intensi nel campo dell'informatica. Esaminiamo qui un semplice schema di ordinamento. Nei Capitoli 12 e 13, analizzeremo schemi più complessi che offrono prestazioni migliori. Spesso gli algoritmi più semplici hanno prestazioni scarse. Il loro pregio è quello di essere facili da scrivere, da testare e da correggere. Per ottenere prestazioni elevate sono spesso necessari algoritmi più complessi.



### Bubble sort

Il programma della Figura 6.12 ordina i valori di un array `a` di 10 elementi (riga 8) in ordine crescente. La tecnica che usiamo è chiamata **bubble sort** (letteralmente “ordinamento a bolle”) o **sinking sort** (letteralmente “ordinamento per affondamento”), perché i valori più piccoli salgono verso la cima dell’array a poco a poco “come bolle”, come le bolle d’aria che si formano nell’acqua, mentre i valori più grandi scendono verso il fondo dell’array. La tecnica consiste nell’effettuare diverse passate lungo l’array. A ogni passata vengono confrontate le successive coppie di elementi (l’elemento 0 e l’elemento 1, poi l’elemento 1 e l’elemento 2, ecc.). Se una coppia è in ordine crescente (o se i valori sono identici), si lasciano i valori come sono. Se una coppia è in ordine decrescente, i valori vengono scambiati nell’array.

```

1 // fig06_12.c
2 // Ordinare i valori di un array in ordine crescente.
3 #include <stdio.h>
4 #define SIZE 10
5
6 // la funzione main inizia l'esecuzione del programma
7 int main(void) {
8 int a[SIZE] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37};
9
10 puts("Data items in original order");
11
12 // stampa l'array originario
13 for (size_t i = 0; i < SIZE; ++i) {
14 printf("%4d", a[i]);
15 }
16
17 // bubble sort

```

```

18 // ciclo per il numero di passate
19 for (int pass = 1; pass < SIZE; ++pass) {
20 // ciclo per il numero di confronti a ogni passata
21 for (size_t i = 0; i < SIZE - 1; ++i) {
22 // confronta due elementi adiacenti e scambiali se il
23 // primo elemento è maggiore del secondo elemento
24 if (a[i] > a[i + 1]) {
25 int hold = a[i];
26 a[i] = a[i + 1];
27 a[i + 1] = hold;
28 }
29 }
30 }
31 puts("\nData items in ascending order");
32
33 // stampa l'array ordinato
34 for (size_t i = 0; i < SIZE; ++i) {
35 printf("%4d", a[i]);
36 }
37
38 puts("");
39
40 }

```

```

Data items in original order
2 6 4 8 10 12 89 68 45 37
Data items in ascending order
2 4 6 8 10 12 37 45 68 89

```

**Figura 6.12** Ordinare i valori di un array in ordine crescente.

Dapprima il programma confronta  $a[0]$  con  $a[1]$ , poi  $a[1]$  con  $a[2]$ , poi  $a[2]$  con  $a[3]$  e così via, finché completa la passata confrontando  $a[8]$  con  $a[9]$ . Benché vi siano 10 elementi, sono eseguiti solo nove confronti. Per via del modo in cui sono fatti i successivi confronti, un valore grande si può muovere in giù lungo l'array di molte posizioni in una singola passata, ma un valore piccolo si può muovere in su solo di una posizione.

Alla prima passata è garantito che il valore più grande scenda giù fino all'elemento che sta al fondo dell'array,  $a[9]$ . Alla seconda passata è garantito che il secondo valore più grande scenda giù fino ad  $a[8]$ . Alla nona passata il nono valore più grande scende fino ad  $a[1]$ . Questo lascia il valore più piccolo in  $a[0]$ , così sono necessarie solo nove passate per ordinare l'array di 10 elementi.

### Scambio di elementi

L'ordinamento è eseguito dai cicli annidati `for` (righe 19-30). Se è necessario uno scambio, questo è eseguito con le tre assegnazioni alle righe 25-27:

```

int hold = a[i];
a[i] = a[i + 1];
a[i + 1] = hold;

```

La variabile `hold` memorizza temporaneamente uno dei due valori da scambiare. Lo scambio non può essere eseguito con le sole due assegnazioni

```

a[i] = a[i + 1];
a[i + 1] = a[i];

```

Se, per esempio,  $a[i]$  è 7 e  $a[i + 1]$  è 5, dopo la prima assegnazione entrambi i valori saranno 5 e il valore 7 sarà perduto. Da qui la necessità di una variabile extra *hold*.

#### Bubble sort: facile da implementare, ma lento

Il principale pregio del bubble sort consiste nella facilità di programmarlo. Tuttavia, esso opera lentamente, poiché ogni scambio sposta un elemento solo di una posizione verso la sua destinazione finale. Questo risulta evidente quando si ordinano array grandi. Negli esercizi esamineremo versioni più efficienti del bubble sort. Sono state sviluppate tecniche di ordinamento di gran lunga più efficienti del bubble sort. Analizzeremo altri algoritmi nel Capitolo 13. I corsi più avanzati di informatica analizzano più approfonditamente l'ordinamento e la ricerca di elementi in array.

#### ✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è *vera*?

- a) Il principale pregio del bubble sort consiste nella facilità di programmarlo.
- b) Il bubble sort opera lentamente, poiché ogni scambio sposta un elemento solo di una posizione verso la sua destinazione finale. Questo risulta evidente quando si ordinano array grandi.
- c) Sono state sviluppate tecniche di ordinamento di gran lunga più efficienti del bubble sort.
- d) Tutte le affermazioni precedenti sono *vere*.

**Risposta:** d.

2. (*Vero/Falso*) Se nel bubble sort è necessario uno scambio, vengono usate le assegnazioni:

```
a[i] = a[i + 1];
a[i + 1] = a[i];
```

**Risposta:** *Falso*. In realtà, lo scambio richiede una variabile in più e un'istruzione in più:

```
int hold = a[i];
a[i] = a[i + 1];
a[i + 1] = hold;
```

dove la variabile *hold* memorizza temporaneamente uno dei due valori da scambiare.

## 6.9 Caso pratico di introduzione alla data science: analisi dei dati di un sondaggio

Consideriamo ora un esempio più ampio. I computer sono comunemente usati per l'**analisi dei dati di sondaggi** per compilare e analizzare i risultati di rilevazioni e indagini demoscopiche. Il programma della Figura 6.13 usa l'array *response* inizializzato con 99 risposte ottenute in un sondaggio. Ogni risposta è un numero da 1 a 9. Il programma calcola la media, la mediana e la moda dei 99 valori. Questo esempio include la maggior parte delle manipolazioni comuni abitualmente richieste nei problemi con array, incluso il passaggio degli array a funzioni. Notate che le righe 48-52 contengono diverse stringhe letterali separate solo da un carattere di spaziatura. Il compilatore del C combina automaticamente tali stringhe letterali in un'unica stringa, e ciò è utile per rendere più leggibili le stringhe lunghe.

```

1 // fig06_13.c
2 // Analisi dei dati di un sondaggio con l'uso di array:
3 // calcolo della media, della mediana e della moda dei dati.
4 #include <stdio.h>
5 #define SIZE 99
6
7 // prototipi di funzioni
8 void mean(const int answer[]);
9 void median(int answer[]);
```

```
10 void mode(int freq[], const int answer[]) ;
11 void bubbleSort(int a[]);
12 void printArray(const int a[]);
13
14 // la funzione main inizia l'esecuzione del programma
15 int main(void) {
16 int frequency[10] = {0}; // inizializza l'array frequency
17
18 // inizializza l'array response
19 int response[SIZE] =
20 {6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
21 7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
22 6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
23 7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
24 6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
25 7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
26 5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
27 7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
28 7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
29 4, 5, 6, 1, 6, 5, 7, 8, 7};
30
31 // elabora le risposte
32 mean(response);
33 median(response);
34 mode(frequency, response);
35 }
36
37 // calcola la media di tutti i valori delle risposte
38 void mean(const int answer[]) {
39 printf("%s\n%s\n%s\n", "-----", " Mean", "-----");
40
41 int total = 0; // variabile per contenere la somma degli elementi dell'array
42
43 // calcola il totale dei valori delle risposte
44 for (size_t j = 0; j < SIZE; ++j) {
45 total += answer[j];
46 }
47
48 printf("The mean is the average value of the data\n"
49 "items. The mean is equal to the total of\n"
50 "all the data items divided by the number\n"
51 "of data items (%u). The mean value for\n"
52 "this run is: %u / %u = %.4f\n\n",
53 SIZE, total, SIZE, (double) total / SIZE);
54 }
55
56 // ordina l'array e determina il valore dell'elemento mediano
57 void median(int answer[]) {
58 printf("\n%s\n%s\n%s\n%s", "-----", " Median", "-----",
59 "The unsorted array of responses is");
60
61 printArray(answer); // stampa l'array non ordinato
62 }
```

```
63 bubbleSort(answer); // ordina l'array
64
65 printf("%s", "\n\nThe sorted array is");
66 printArray(answer); // stampa l'array ordinato
67
68 // stampa l'elemento mediano
69 printf("\n\nThe median is element %u of\n"
70 "the sorted %u element array.\n"
71 "For this run the median is %u\n\n",
72 SIZE / 2, SIZE, answer[SIZE / 2]);
73 }
74
75 // determina la risposta più frequente
76 void mode(int freq[], const int answer[]) {
77 printf("\n%*s\n%*s\n%*s\n", "-----", " Mode", "-----");
78
79 // inizializza le frequenze a 0
80 for (size_t rating = 1; rating <= 9; ++rating) {
81 freq[rating] = 0;
82 }
83
84 // calcola le frequenze
85 for (size_t j = 0; j < SIZE; ++j) {
86 ++freq[answer[j]];
87 }
88
89 // stampa le intestazioni per le colonne dei risultati
90 printf("%s%11s%19s\n\n%54s\n%54s\n\n",
91 "Response", "Frequency", "Bar Chart",
92 "1 1 2 2", "5 0 5 0 5");
93
94 // stampa i risultati
95 int largest = 0; // rappresenta la frequenza maggiore
96 int modeValue = 0; // rappresenta la risposta più frequente
97
98 for (size_t rating = 1; rating <= 9; ++rating) {
99 printf("%8zu%11d ", rating, freq[rating]);
100
101 // cerca la moda e la frequenza maggiore
102 if (freq[rating] > largest) {
103 largest = freq[rating];
104 modeValue = rating;
105 }
106
107 // stampa la barra che rappresenta il valore della frequenza
108 for (int h = 1; h <= freq[rating]; ++h) {
109 printf("%s", "*");
110 }
111
112 puts(""); // inizia una nuova riga di stampa
113 }
114
115 // stampa il valore della moda
```

```

116 printf("\nThe mode is the most frequent value.\n"
117 "For this run the mode is %d which occurred %d times.\n",
118 modeValue, largest);
119 }
120
121 // funzione che ordina un array con l'algoritmo bubble sort
122 void bubbleSort(int a[]) {
123 // ciclo per il numero di passate
124 for (int pass = 1; pass < SIZE; ++pass) {
125 // ciclo per il numero di confronti a ogni passata
126 for (size_t j = 0; j < SIZE - 1; ++j) {
127 // scambia gli elementi se non sono in ordine
128 if (a[j] > a[j + 1]) {
129 int hold = a[j];
130 a[j] = a[j + 1];
131 a[j + 1] = hold;
132 }
133 }
134 }
135 }
136
137 // stampa i contenuti dell'array (20 valori per riga)
138 void printArray(const int a[]) {
139 // stampa i contenuti dell'array
140 for (size_t j = 0; j < SIZE; ++j) {
141
142 if (j % 20 == 0) { // inizia una nuova riga ogni 20 valori
143 puts("");
144 }
145
146 printf("%2d", a[j]);
147 }
148 }

```

-----  
Mean

The mean is the average value of the data items. The mean is equal to the total of all the data items divided by the number of data items (99). The mean value for this run is: 681 / 99 = 6.8788

-----  
Median

The unsorted array of responses is  
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8  
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9  
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3  
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8  
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

```
The sorted array is
1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
5 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

The median is element 49 of  
the sorted 99 element array.  
For this run the median is 7

#### Mode

| Response | Frequency | Bar Chart |
|----------|-----------|-----------|
| 1        | 1         | *         |
| 2        | 3         | ***       |
| 3        | 4         | ****      |
| 4        | 5         | *****     |
| 5        | 8         | *****     |
| 6        | 9         | *****     |
| 7        | 23        | *****     |
| 8        | 27        | *****     |
| 9        | 19        | *****     |

The mode is the most frequent value.  
For this run the mode is 8 which occurred 27 times.

**Figura 6.13** Analisi dei dati di un sondaggio con l'uso di array: calcolo della media, della mediana e della moda dei dati.

#### Media

La *media* è la media aritmetica dei 99 valori. La funzione `mean` (righe 38-54) calcola la media sommando i 99 elementi e dividendo il risultato per 99.

#### Mediana

La mediana è il valore di mezzo. La funzione `median` (righe 57-73) dapprima ordina le risposte chiamando la funzione `bubbleSort` (definita nelle righe 122-135). Poi determina la mediana selezionando l'elemento centrale dell'array ordinato, `answer[SIZE / 2]`. Quando il numero di elementi è pari, la mediana deve essere calcolata come la media dei due elementi centrali. La funzione `median` non effettua in realtà questo ulteriore calcolo. Le righe 61 e 66 chiamano la funzione `printArray` (righe 138-148) che stampa l'array `response` prima e dopo l'ordinamento.

#### Moda

La moda è il valore che ricorre più frequentemente tra le 99 risposte. La funzione `mode` (righe 76-119) determina la moda contando il numero di risposte di ogni tipo e selezionando il valore con il conteggio maggiore. Questa versione della funzione `mode` non si occupa delle situazioni di parità (vedi Esercizio 6.14). La funzione `mode` produce anche un grafico a barre per permettere di determinare graficamente la moda.

**Esercizi correlati**

Questo caso pratico è affiancato dal seguente esercizio:

- Esercizio 6.14 (Modifiche al programma per media, mediana e moda).

**✓ Autovalutazione**

1. *(Scelta multipla)* Quale delle seguenti affermazioni a), b) o c) è *falsa*?
  - a) La mediana è il valore centrale in una serie ordinata di dati.
  - b) L'algoritmo che determina la mediana dei valori in un array ordina l'array in ordine crescente e poi seleziona l'elemento centrale dell'array ordinato.
  - c) Quando il numero di elementi è pari, la mediana viene calcolata come la media dei due elementi centrali.
  - d) Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

2. *(Scelta multipla)* Quale delle seguenti affermazioni a), b) o c) è *falsa*?
  - a) La moda è il valore che ricorre più frequentemente in un insieme di dati.
  - b) L'algoritmo che determina la moda conta il numero di occorrenze di ciascun valore e poi seleziona il valore con il maggior numero di occorrenze. Un problema che si presenta quando si determina la moda è cosa fare in caso di parità.
  - c) La moda può essere determinata in modo visuale rappresentando graficamente le frequenze dei valori in un grafico a barre: la barra più lunga rappresenta la moda.
  - d) Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

## 6.10 Ricerca in array

Spesso lavorerete con una grande quantità di dati memorizzati negli array. Può essere necessario determinare se un array contiene un valore che corrisponda a un certo **valore chiave**. Il processo che permette di trovare un valore chiave in un array è chiamato **ricerca**. In questo paragrafo esaminiamo due tecniche di ricerca: la tecnica semplice di **ricerca lineare** e la tecnica più efficiente (ma più complessa) di **ricerca binaria**. Gli Esercizi 6.32 e 6.33 vi chiedono di implementare le versioni ricorsive rispettivamente della ricerca lineare e della ricerca binaria.

### 6.10.1 Effettuare una ricerca in un array con la ricerca lineare

Una ricerca lineare (Figura 6.14, righe 37-46) confronta ogni elemento dell'array con la **chiave di ricerca**. Dal momento che l'array non è in un ordine particolare, è altrettanto probabile che il valore venga trovato nel primo elemento quanto nell'ultimo. In media, dunque, il programma dovrà confrontare la chiave di ricerca con *metà* degli elementi dell'array. Se il valore chiave viene trovato, si restituisce l'indice dell'elemento; in caso contrario, si restituisce -1 (un indice non valido).

```

1 // fig06_14.c
2 // Ricerca lineare in un array.
3 #include <stdio.h>
4 #define SIZE 100
5
6 // prototipo di funzione
7 int linearSearch(const int array[], int key, size_t size);
8
9 // la funzione main inizia l'esecuzione del programma
10 int main(void) {
11 int a[SIZE] = {0}; // crea l'array a
12

```

```

13 // crea alcuni dati
14 for (size_t x = 0; x < SIZE; ++x) {
15 a[x] = 2 * x;
16 }
17
18 printf("Enter integer search key: ");
19 int searchKey = 0; // valore da localizzare nell'array a
20 scanf("%d", &searchKey);
21
22 // tenta di localizzare searchKey nell'array a
23 int subscript = linearSearch(a, searchKey, SIZE);
24
25 // stampa i risultati
26 if (subscript != -1) {
27 printf("Found value at subscript %d\n", subscript);
28 }
29 else {
30 puts("Value not found");
31 }
32 }
33
34 // confronta la chiave con ogni elemento dell'array fino a quando non viene
35 // trovata la posizione o raggiunta la fine dell'array; restituisci l'indice
36 // dell'elemento se la chiave viene trovata o -1 se la chiave non viene trovata
37 int linearSearch(const int array[], int key, size_t size) {
38 // ciclo attraverso l'array
39 for (size_t n = 0; n < size; ++n) {
40 if (array[n] == key) {
41 return n; // restituisci la posizione della chiave
42 }
43 }
44
45 return -1; // chiave non trovata
46 }

```

Enter integer search key: 36  
Found value at subscript 18

Enter integer search key: 37  
Value not found

**Figura 6.14** Ricerca lineare in un array.

### 6.10.2 Effettuare una ricerca in un array con la ricerca binaria

Il metodo della ricerca lineare funziona bene per array piccoli o non ordinati. Tuttavia, per array grandi la ricerca lineare è inefficiente. Se l'array è ordinato, si può usare la tecnica molto veloce di ricerca binaria.

L'algoritmo di ricerca binaria elimina dalla ricerca *una metà* degli elementi di un array ordinato dopo ogni confronto. L'algoritmo localizza l'elemento centrale dell'array e lo confronta con la chiave di ricerca. Se sono uguali, l'algoritmo ha trovato la chiave di ricerca, quindi restituisce l'indice di quell'elemento. Se non sono uguali, il problema si riduce alla ricerca in *una metà* dell'array. Se la chiave di ricerca è minore dell'elemento centrale dell'array, l'algoritmo effettua una ricerca nella *prima metà* dell'array; altrimenti la effettua nella *se-*

*seconda metà.* Se la chiave di ricerca non è l'elemento centrale del sottoarray corrente (una porzione dell'array originario), l'algoritmo viene ripetuto su un quarto dell'array originario. La ricerca continua finché la chiave di ricerca non è uguale all'elemento centrale di un sottoarray, o finché tale sottoarray non è costituito da un solo elemento che non è uguale alla chiave di ricerca (ossia la chiave di ricerca non è stata trovata).

#### • Prestazioni dell'algoritmo di ricerca binaria

Nel peggior dei casi, effettuare una ricerca in un array di 1023 elementi richiede soltanto 10 confronti con la ricerca binaria. Dividere ripetutamente 1024 per 2 produce i valori 512, 256, 128, 64, 32, 16, 8, 4, 2 e 1. Il numero 1024 ( $2^{10}$ ) viene diviso per due solo 10 volte per ottenere il valore 1. Ogni divisione per 2 corrisponde a un confronto nell'algoritmo di ricerca binaria. Un array di 1.048.576 ( $2^{20}$ ) elementi richiede un massimo di soli 20 confronti per trovare la chiave di ricerca. Un array ordinato di un miliardo di elementi richiede un massimo di soli 30 confronti per trovare la chiave di ricerca. Questo è un miglioramento straordinario delle prestazioni rispetto alla ricerca lineare di un array ordinato, che richiede il confronto della chiave di ricerca in media con la metà degli elementi di un array. Per un array di un miliardo di elementi si ha una differenza tra una media di 500 milioni di confronti e un massimo di 30 confronti! Il massimo numero di confronti per qualunque array può essere determinato considerando la prima potenza di 2 maggiore del numero degli elementi dell'array.

#### Implementazione della ricerca binaria

Il programma della Figura 6.15 presenta la versione iterativa della funzione `binarySearch` (righe 39-60). La funzione riceve quattro argomenti: un array intero `b` su cui effettuare la ricerca, un intero `key` da trovare, l'indice di array `low` e l'indice di array `high`. Gli ultimi due argomenti definiscono la porzione dell'array su cui effettuare la ricerca. Se la chiave di ricerca non corrisponde all'elemento centrale del sottoarray, l'indice `low` o l'indice `high` viene modificato così da effettuare la ricerca in un sottoarray ancora più piccolo:

- se la chiave di ricerca è minore dell'elemento centrale, l'algoritmo imposta l'indice `high` a `middle - 1` (riga 52), poi continua la ricerca sugli elementi con indici compresi tra `low` e `middle - 1`;
- se la chiave di ricerca è maggiore dell'elemento centrale, l'algoritmo imposta l'indice `low` a `middle + 1` (riga 55), poi continua la ricerca sugli elementi con indici compresi tra `middle + 1` e `high`.

Il programma usa un array di 15 elementi. La prima potenza di 2 maggiore del numero degli elementi in questo array è 16 ( $2^4$ ), così non ci vogliono più di 4 confronti per trovare la chiave di ricerca. Il programma usa la funzione `printHeader` (righe 63-79) per stampare gli indici dell'array e la funzione `printRow` (righe 83-99) per stampare ogni sottoarray durante il processo di ricerca binaria. L'elemento centrale in ogni sottoarray è segnato con un asterisco (\*) per indicare l'elemento con il quale viene confrontata la chiave di ricerca.

```

1 // fig06_15.c
2 // Ricerca binaria in un array ordinato.
3 #include <stdio.h>
4 #define SIZE 15
5
6 // prototipi di funzioni
7 int binarySearch(const int b[], int key, size_t low, size_t high);
8 void printHeader(void);
9 void printRow(const int b[], size_t low, size_t mid, size_t high);
10
11 // la funzione main inizia l'esecuzione del programma
12 int main(void) {
13 int a[SIZE] = {0}; // crea l'array a
14
15 // crea i dati
16 for (size_t i = 0; i < SIZE; ++i) {
17 a[i] = 2 * i;
18 }
19 }
```

```
20 printf("%s", "Enter a number between 0 and 28: ");
21 int key = 0; // valore da localizzare nell'array a
22 scanf("%d", &key);
23
24 printHeader();
25
26 // cerca la chiave nell'array a
27 int result = binarySearch(a, key, 0, SIZE - 1);
28
29 // stampa i risultati
30 if (result != -1) {
31 printf("\n%d found at subscript %d\n", key, result);
32 }
33 else {
34 printf("\n%d not found\n", key);
35 }
36 }
37
38 // funzione che esegue la ricerca binaria su un array
39 int binarySearch(const int b[], int key, size_t low, size_t high) {
40 // ripeti finche' l'indice low non diventa maggiore dell'indice high
41 while (low <= high) {
42 size_t middle = (low + high) / 2; // determina l'indice middle
43
44 // stampa il sottoarray considerato in questa iterazione del ciclo
45 printRow(b, low, middle, high);
46
47 // se key corrisponde, restituisci l'indice middle
48 if (key == b[middle]) {
49 return middle;
50 }
51 else if (key < b[middle]) { // se key < b[middle], modifica high
52 high = middle - 1; // cerca nella parte bassa dell'array
53 }
54 else { // se key > b[middle], modifica low
55 low = middle + 1; // cerca nella parte alta dell'array
56 }
57 } // fine di while
58
59 return -1; // searchKey non trovato
60 }
61
62 // Stampa l'intestazione per l'output
63 void printHeader(void) {
64 puts("\nSubscripts:");
65
66 // stampa l'intestazione della colonna
67 for (int i = 0; i < SIZE; ++i) {
68 printf("%3d ", i);
69 }
70
71 puts(""); // inizia una nuova riga di stampa
72 }
```

```

73 // stampa una riga di caratteri -
74 for (int i = 1; i <= 4 * SIZE; ++i) {
75 printf("%s", "-");
76 }
77
78 puts(""); // inizia una nuova riga di stampa
79 }
80
81 // Stampa una riga dell'output che mostra la parte
82 // corrente dell'array ancora da processare.
83 void printRow(const int b[], size_t low, size_t mid, size_t high) {
84 // ciclo attraverso l'intero array
85 for (size_t i = 0; i < SIZE; ++i) {
86 // stampa spazi se si e' al di fuori del sottoarray corrente
87 if (i < low || i > high) {
88 printf("%s", " ");
89 }
90 else if (i == mid) { // stampa l'elemento centrale
91 printf("%3d*", b[i]); // marca il valore centrale
92 }
93 else { // stampa gli altri elementi del sottoarray
94 printf("%3d ", b[i]);
95 }
96 }
97
98 puts(""); // inizia una nuova riga di stampa
99 }
```

Enter a number between 0 and 28: 25

Subscripts:

|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|

---

|   |   |   |   |   |    |    |     |    |    |    |     |     |    |    |
|---|---|---|---|---|----|----|-----|----|----|----|-----|-----|----|----|
| 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14* | 16 | 18 | 20 | 22  | 24  | 26 | 28 |
|   |   |   |   |   |    |    |     | 16 | 18 | 20 | 22* | 24  | 26 | 28 |
|   |   |   |   |   |    |    |     |    |    |    | 24  | 26* | 28 |    |
|   |   |   |   |   |    |    |     |    |    |    |     | 24* |    |    |

25 not found

Enter a number between 0 and 28: 8

Subscripts:

|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|

---

|   |   |   |    |   |    |    |     |     |    |    |    |    |    |    |
|---|---|---|----|---|----|----|-----|-----|----|----|----|----|----|----|
| 0 | 2 | 4 | 6  | 8 | 10 | 12 | 14* | 16  | 18 | 20 | 22 | 24 | 26 | 28 |
| 0 | 2 | 4 | 6* | 8 | 10 | 12 |     |     |    |    |    |    |    |    |
|   |   |   |    |   |    |    | 8   | 10* | 12 |    |    |    |    |    |
|   |   |   |    |   |    |    |     | 8*  |    |    |    |    |    |    |

8 found at subscript 4

```

Enter a number between 0 and 28: 6
Subscripts:
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

 0 2 4 6 8 10 12 14* 16 18 20 22 24 26 28

 0 2 4 6* 8 10 12
6 found at subscript 3

```

**Figura 6.15** Ricerca binaria in un array ordinato.

### ✓ Autovalutazione

1. *(Scelta multipla)* Quale delle seguenti affermazioni sulla ricerca lineare è *falsa*?
    - a) La ricerca lineare confronta ogni elemento dell'array con la chiave di ricerca.
    - b) Dal momento che l'array non è in un ordine particolare, è altrettanto probabile che il valore venga trovato nel primo elemento quanto nell'ultimo.
    - c) In media la ricerca lineare confronta la chiave di ricerca con metà degli elementi dell'array.
    - d) La ricerca lineare funziona bene per array piccoli o non ordinati. Per array *grandi* è *inefficiente*.
- Risposta:** a) è *falsa*. In realtà, la ricerca lineare potrebbe trovare una corrispondenza prima di raggiungere la fine dell'array, e in tal caso terminare prima di aver confrontato ogni elemento con la chiave di ricerca.
2. *(Scelta multipla)* Quale delle seguenti affermazioni sulla ricerca binaria è *falsa*?
    - a) Se l'array è ordinato, si può usare la tecnica molto veloce di ricerca binaria.
    - b) L'algoritmo di ricerca binaria elimina dalla ricerca due degli elementi di un array ordinato dopo ogni confronto.
    - c) L'algoritmo localizza l'elemento centrale dell'array e lo confronta con la chiave di ricerca. Se sono uguali, la chiave di ricerca è stata trovata e viene restituito l'indice di array di quell'elemento. Se la chiave di ricerca è minore dell'elemento centrale dell'array, l'algoritmo effettua una ricerca nella prima metà dell'array, altrimenti la effettua nella seconda metà.
    - d) La ricerca continua finché la chiave di ricerca non è uguale all'elemento centrale di un sottoarray o finché tale sottoarray non è costituito da un solo elemento che non è uguale alla chiave di ricerca (ossia la chiave di ricerca non viene trovata).
- Risposta:** b) è *falsa*. In realtà, l'algoritmo di ricerca binaria elimina dalla ricerca una metà degli elementi di un array ordinato dopo ogni confronto.

## 6.11 Array multidimensionali

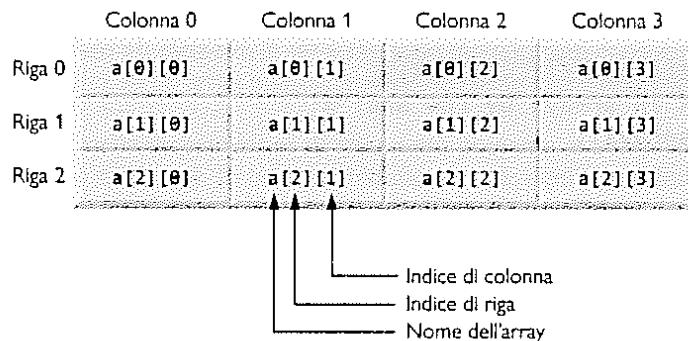
Gli array possono avere diversi indici. Un uso comune di questi **array multidimensionali** è quello di rappresentare tabelle di valori che contengono informazioni disposte in *righe* e *colonne*. Per identificare un elemento particolare di una tabella, dobbiamo specificare due indici:

- il *primo* identifica (per convenzione) la *riga* dell'elemento;
- il *secondo* identifica (per convenzione) la *colonna* dell'elemento.

Gli array che richiedono due indici per identificare un elemento specifico sono chiamati comunemente **array bidimensionali**. Gli array multidimensionali possono avere più di due indici.

### 6.11.1 Illustrare un array bidimensionale

Il seguente diagramma illustra un array bidimensionale chiamato a:



L'array contiene tre righe e quattro colonne, per cui è detto array 3-per-4. In generale, un array con  $m$  righe e  $n$  colonne è chiamato **array m-per-n**.

Ogni elemento nell'array a è identificato da un nome della forma  $a[i][j]$ , dove a è il nome dell'array e i e j sono gli indici che identificano unicamente ogni elemento. I nomi degli elementi nella riga 0 hanno tutti il primo indice uguale a 0. I nomi degli elementi nella colonna 3 hanno tutti il secondo indice uguale a 3. Fare riferimento all'elemento di un array bidimensionale con  $a[x, y]$  anziché con  $a[x][y]$  è un errore logico.

X Il C tratta  $a[x, y]$  come  $a[y]$ , per cui questo errore di programmazione non è un errore di *sintassi*. La virgola in questo contesto è un **operatore virgola** che garantisce che le sequenze di espressioni vengano calcolate da sinistra a destra. Il valore di una sequenza di espressioni separate da virgolette è quello dell'espressione più a destra.

### 6.11.2 Inizializzare un array con due indici

Potete inizializzare un array multidimensionale quando lo definite. Per esempio, potete definire e inizializzare l'array bidimensionale int b[2][2] con:

```
int b[2][2] = {{1, 2}, {3, 4}};
```

I valori nella lista di inizializzatori sono raggruppati per riga in parentesi graffe. I valori nella prima coppia di parentesi graffe inizializzano la riga 0 e i valori nella seconda coppia di parentesi graffe inizializzano la riga 1. Così i valori 1 e 2 inizializzano rispettivamente gli elementi  $b[0][0]$  e  $b[0][1]$ , e i valori 3 e 4 inizializzano rispettivamente gli elementi  $b[1][0]$  e  $b[1][1]$ . Se non vi sono abbastanza inizializzatori per una data riga, i restanti elementi di quella riga sono inizializzati a 0. Pertanto, la definizione:

```
int b[2][2] = {{1}, {3, 4}};
```

inizializza  $b[0][0]$  a 1,  $b[0][1]$  a 0,  $b[1][0]$  a 3 e  $b[1][1]$  a 4. Il programma della Figura 6.16 illustra la definizione e l'inizializzazione di array bidimensionali.

```

1 // fig06_16.c
2 // Inizializzazione di array multidimensionali.
3 #include <stdio.h>
4
5 void printArray(int a[][3]); // prototipo di funzione
6
7 // la funzione main inizia l'esecuzione del programma
8 int main(void) {
9 int array1[2][3] = {{1, 2, 3}, {4, 5, 6}};
10 puts("Values in array1 by row are:");

```

```

11 printArray(array1);
12
13 int array2[2][3] = {{1, 2, 3}, {4, 5}};
14 puts("Values in array2 by row are:");
15 printArray(array2);
16
17 int array3[2][3] = {{1, 2}, {4}};
18 puts("Values in array3 by row are:");
19 printArray(array3);
20 }
21
22 // funzione per stampare un array con due righe e tre colonne
23 void printArray(int a[][3]) {
24 // iterazione per righe
25 for (size_t i = 0; i <= 1; ++i) {
26 // stampa i valori delle colonne
27 for (size_t j = 0; j <= 2; ++j) {
28 printf("%d ", a[i][j]);
29 }
30
31 printf("\n"); // inizia una nuova riga di stampa
32 }
33 }
```

Values in array1 by row are:

1 2 3  
4 5 6

Values in array2 by row are:

1 2 3  
4 5 0

Values in array3 by row are:

1 2 0  
4 0 0

**Figura 6.16** Inizializzazione di array multidimensionali.

#### Definizione di array1

Il programma definisce tre array di due righe e tre colonne. La definizione di `array1` (riga 9) fornisce sei inizializzatori in due sottoliste. La prima sottolista inizializza la *riga 0* ai valori 1, 2 e 3, e la seconda sottolista inizializza la *riga 1* ai valori 4, 5 e 6.

#### Definizione di array2

La definizione di `array2` (riga 13) fornisce cinque inizializzatori in due sottoliste, inizializzando la *riga 0* a 1, 2 e 3, e la *riga 1* a 4, 5 e 0. Gli elementi che non hanno un inizializzatore esplicito sono inizializzati automaticamente a zero, così `array2[1][2]` è inizializzato a 0.

#### Definizione di array3

La definizione di `array3` (riga 17) fornisce tre inizializzatori in due sottoliste. La sottolista per la prima riga inizializza esplicitamente i primi due elementi della prima riga a 1 e a 2 e inizializza implicitamente il terzo elemento a 0. La sottolista per la seconda riga inizializza esplicitamente il primo elemento a 4 e inizializza implicitamente gli ultimi due elementi a 0.

### Funzione printArray

Il programma chiama `printArray` (righe 23-33) per stampare gli elementi di ogni array. La definizione della funzione specifica il parametro array come `int a[][]`. In un parametro array unidimensionale, le parentesi dell'array sono vuote. Il primo indice di un array multidimensionale non è neppure necessario, mentre lo sono tutti gli indici successivi. Il compilatore usa questi indici per determinare le posizioni in memoria degli elementi di array multidimensionali. Tutti gli elementi di un array sono memorizzati consecutivamente nella memoria a prescindere dal numero di indici. In un array bidimensionale, la prima riga è memorizzata in memoria seguita dalla seconda.

Fornire i valori degli indici nella dichiarazione di parametro permette al compilatore di dire alla funzione come localizzare ogni elemento nell'array. In un array bidimensionale ogni riga è fondamentalmente un array unidimensionale. Per localizzare un elemento in una riga specifica, il compilatore deve sapere quanti elementi vi sono in ogni riga, così che possa saltare il giusto numero di locazioni di memoria quando accede all'array. In questo modo, quando accede a `a[1][2]` nel nostro esempio, il compilatore sa che deve saltare i tre elementi della prima riga per raggiungere la seconda riga (riga 1). Quindi, il compilatore accede all'elemento 2 di quella riga.

### 6.11.3 Impostare gli elementi in una riga

Molte manipolazioni comuni di array usano istruzioni di iterazione `for`. Per esempio, l'istruzione seguente imposta a zero tutti gli elementi nella riga 2 dell'array 3-per-4 a di tipo `int`:

```
for (int column = 0; column <= 3; ++column) {
 a[2][column] = 0;
}
```

Abbiamo specificato la riga 2, così il primo indice è sempre 2. Il ciclo fa variare soltanto l'indice di colonna. La precedente istruzione `for` è equivalente alle istruzioni di assegnazione:

```
a[2][0] = 0;
a[2][1] = 0;
a[2][2] = 0;
a[2][3] = 0;
```

### 6.11.4 Calcolare il totale degli elementi in un array bidimensionale

La seguente istruzione `for` annidata calcola il totale degli elementi nell'array 3 per 4 a di tipo `int`:

```
int total = 0;

for (int row = 0; row <= 2; ++row) {
 for (int column = 0; column <= 3; ++column) {
 total += a[row][column];
 }
}
```

L'istruzione `for` calcola il totale degli elementi una riga alla volta. L'istruzione `for` esterna inizia impostando l'indice di `row` a 0 in modo che gli elementi di quella riga possono essere sommati dall'istruzione `for` interna. L'istruzione `for` esterna poi incrementa `row` a 1, in modo che gli elementi di quella riga possono essere sommati. Quindi, l'istruzione `for` esterna incrementa `row` a 2, in modo che gli elementi di quella riga possono essere sommati. Quando l'istruzione `for` annidata termina, `total` contiene la somma di tutti gli elementi dell'array `a`.

### 6.11.5 Manipolazioni di array bidimensionali

Il programma della Figura 6.17 usa istruzioni `for` per eseguire diverse manipolazioni comuni sull'array 3-per-4 `studentGrades`. Ogni riga rappresenta uno studente, e ogni colonna rappresenta un voto a uno dei quattro esami che gli studenti hanno sostenuto durante il semestre. Le manipolazioni dell'array sono eseguite da quattro funzioni:

- La funzione `minimum` (righe 38-52) trova il voto più basso fra quelli di tutti gli studenti nel semestre.
- La funzione `maximum` (righe 55-69) trova il voto più alto fra quelli di tutti gli studenti nel semestre.
- La funzione `average` (righe 72-81) calcola la media nel semestre di un particolare studente.
- La funzione `printArray` (righe 84-98) stampa l'array bidimensionale in un formato tabellare ordinato.

```
1 // fig06_17.c
2 // Manipolazioni di un array bidimensionale.
3 #include <stdio.h>
4 #define STUDENTS 3
5 #define EXAMS 4
6
7 // prototipi di funzioni
8 int minimum(const int grades[][EXAMS], size_t pupils, size_t tests);
9 int maximum(const int grades[][EXAMS], size_t pupils, size_t tests);
10 double average(const int setOfGrades[], size_t tests);
11 void printArray(const int grades[][EXAMS], size_t pupils, size_t tests);
12
13 // la funzione main inizia l'esecuzione del programma
14 int main(void) {
15 // inizializza i voti per i tre studenti (righe)
16 int studentGrades[STUDENTS][EXAMS] =
17 {{77, 68, 86, 73},
18 {96, 87, 89, 78},
19 {70, 90, 86, 81}};
20
21 // stampa l'array studentGrades
22 puts("The array is:");
23 printArray(studentGrades, STUDENTS, EXAMS);
24
25 // determina i valori minimo e massimo dei voti
26 printf("\n\nLowest grade: %d\nHighest grade: %d\n",
27 minimum(studentGrades, STUDENTS, EXAMS),
28 maximum(studentGrades, STUDENTS, EXAMS));
29
30 // calcola la media dei voti per ogni studente
31 for (size_t student = 0; student < STUDENTS; ++student) {
32 printf("The average grade for student %zu is %.2f\n",
33 student, average(studentGrades[student], EXAMS));
34 }
35 }
36
37 // Trova il voto minimo
38 int minimum(const int grades[][EXAMS], size_t pupils, size_t tests) {
39 int lowGrade = 100; // inizializza al voto più alto possibile
40
41 // ciclo per le righe di grades
42 for (size_t row = 0; row < pupils; ++row) {
43 // ciclo per le colonne di grades
44 for (size_t column = 0; column < tests; ++column) {
45 if (grades[row][column] < lowGrade) {
46 lowGrade = grades[row][column];
47 }
48 }
49 }
50 }
```

```
48 }
49 }
50 return lowGrade; // restituisci il voto minimo
51 }
52 }
53 // Trova il voto massimo
54 int maximum(const int grades[][EXAMS], size_t pupils, size_t tests) {
55 int highGrade = 0; // inizializza al voto più basso possibile
56
57 // ciclo per le righe di grades
58 for (size_t row = 0; row < pupils; ++row) {
59 // ciclo per le colonne di grades
60 for (size_t column = 0; column < tests; ++column) {
61 if (grades[row][column] > highGrade) {
62 highGrade = grades[row][column];
63 }
64 }
65 }
66 }
67 return highGrade; // restituisci il voto massimo
68 }
69 }
70
71 // Determina il voto medio per un particolare studente
72 double average(const int setOfGrades[], size_t tests) {
73 int total = 0; // somma dei voti degli esami
74
75 // totale di tutti i voti per uno studente
76 for (size_t test = 0; test < tests; ++test) {
77 total += setOfGrades[test];
78 }
79
80 return (double) total / tests; // media
81 }
82
83 // Stampa l'array
84 void printArray(const int grades[][EXAMS], size_t pupils, size_t tests) {
85 // stampa le intestazioni delle colonne
86 printf("%s", " [0] [1] [2] [3]");
87
88 // stampa i voti in formato tabellare
89 for (size_t row = 0; row < pupils; ++row) {
90 // stampa l'etichetta per la riga
91 printf("\nstudentGrades[%zu] ", row);
92
93 // stampa i voti per uno studente
94 for (size_t column = 0; column < tests; ++column) {
95 printf("%-5d", grades[row][column]);
96 }
97 }
98 }
```

```
The array is:
studentGrades[0] [0] 77 68 86 73
studentGrades[1] 96 87 89 78
studentGrades[2] 70 90 86 81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

**Figura 6.17** Manipolazioni di un array bidimensionale.**Cicli annidati nelle funzioni minimum, maximum e printArray**

Le funzioni `minimum`, `maximum` e `printArray` ricevono ciascuna tre argomenti: l'array `studentGrades` (chiamato `grades` in ogni funzione), il numero di studenti (righe nell'array) e il numero di esami (colonne nell'array). Ogni funzione effettua un ciclo per tutto l'array `grades` usando istruzioni `for` annidate. La seguente istruzione `for` annidata è tratta dalla definizione della funzione `minimum`:

```
// ciclo per le righe di grades
for (size_t row = 0; row < pupils; ++row) {
 // ciclo per le colonne di grades
 for (size_t column = 0; column < tests; ++column) {
 if (grades[row][column] < lowGrade) {
 lowGrade = grades[row][column];
 }
 }
}
```

L'istruzione `for` esterna inizia impostando `row` a 0 in modo che gli elementi di quella riga (cioè i voti del primo studente) si possano confrontare con la variabile `lowGrade` nell'istruzione `for` interna. L'istruzione `for` interna effettua un'iterazione per i quattro voti di una riga specifica e confronta ogni voto con `lowGrade`. Se un voto è minore di `lowGrade`, l'istruzione `if` annidata imposta `lowGrade` a quel voto. L'istruzione `for` esterna incrementa quindi `row` a 1, e gli elementi di quella riga sono confrontati con `lowGrade`. L'istruzione `for` esterna incrementa quindi `row` a 2, e gli elementi di quella riga sono confrontati con `lowGrade`. Quando l'esecuzione dell'istruzione annidata è completa, `lowGrade` contiene il voto più piccolo nell'array bidimensionale. La funzione `maximum` opera in modo simile alla funzione `minimum`.

**Funzione average**

La funzione `average` (righe 72-81) prende due argomenti: un array unidimensionale dei voti degli esami per un particolare studente (`setOfGrades`) e il numero dei voti nell'array. Quando la riga 33 chiama la funzione `average`, il primo argomento – `studentGrades[student]` – passa l'indirizzo di una riga dell'array bidimensionale. L'argomento `studentGrades[1]` è l'indirizzo di partenza della riga 1 dell'array. Ricordate che un array bidimensionale è fondamentalmente un array di array unidimensionali e che il nome di un array unidimensionale è l'indirizzo dell'array in memoria. La funzione `average` calcola la somma degli elementi dell'array, divide il totale per il numero dei voti e restituisce il risultato in virgola mobile.

**✓ Autovalutazione**

- (Cosa fa questo codice?)* Cosa fa la seguente istruzione `for` annidata?

```
product = 1;

for (row = 0; row <= 2; ++row) {
```

```

 for (column = 0; column <= 3; ++column) {
 product *= m[row][column];
 }
}
}

```

**Risposta:** Calcola il prodotto di tutti i valori degli elementi nell'array 3-per-4 *m* di tipo double.

2. (*Cosa fa questo codice?*) Cosa fa la seguente istruzione for annidata?

```

// ciclo per le righe di grades
for (i = 0; i < pupils; ++i) {
 // ciclo per le colonne di grades
 for (j = 0; j < tests; ++j) {
 if (grades[i][j] < lowGrade) {
 lowGrade = grades[i][j];
 }
 }
}

```

**Risposta:** Effettua un ciclo per tutto l'array bidimensionale *grades*, che ha come righe *pupils* e come colonne *tests*, cercando di trovare il voto minimo nell'array. Assumendo che i voti vadano da 0 a 100, *lowGrade* deve essere inizializzato a un valore uguale o maggiore di 100.

## 6.12 Array di lunghezza variabile

Per ogni array che avete definito fin qui, ne avete specificato la dimensione al momento della compilazione. Ma cosa succede se non potete determinare la dimensione di un array fino al momento dell'esecuzione? In passato, per trattare questo caso si doveva usare l'allocazione dinamica di memoria (introdotta nel Capitolo 12). Nei casi in cui la dimensione di un array non è nota al momento della compilazione, C dispone di **array di lunghezza variabile** (VLA, *variable-length array*), ovvero array con lunghezze determinate da espressioni calcolate al momento dell'esecuzione.<sup>2</sup> Il programma della Figura 6.18 dichiara e stampa diversi VLA.

```

1 // fig06_18.c
2 // Uso di array di lunghezza variabile in C99
3 #include <stdio.h>
4
5 // prototipi di funzioni
6 void print1DArray(size_t size, int array[size]);
7 void print2DArray(size_t row, size_t col, int array[row][col]);
8
9 int main(void) {
10 printf("%s", "Enter size of a one-dimensional array: ");
11 int arraySize = 0; // dimensione di un array 1-D
12 scanf("%d", &arraySize);
13
14 int array[arraySize]; // dichiara array 1-D di lunghezza variabile
15
16 printf("%s", "Enter number of rows and columns in a 2-D array: ");
17 int row1 = 0; // numero di righe in un array 2-D
18 int col1 = 0; // numero di colonne in un array 2-D
19 scanf("%d %d", &row1, &col1);
20
21 int array2D1[row1][col1]; // dichiara array 2-D di lunghezza variabile
22

```

2. Questa funzionalità non è supportata da Visual C++ di Microsoft.

```
23 printf("%s",
24 "Enter number of rows and columns in another 2-D array: ");
25 int row2 = 0; // numero di righe in un array 2-D
26 int col2 = 0; // numero di colonne in un array 2-D
27 scanf("%d %d", &row2, &col2);
28
29 int array2D2[row2][col2]; // dichiara array 2-D di lunghezza variabile
30
31 // test dell'operatore sizeof su VLA
32 printf("\nsizeof(array) yields array size of %zu bytes\n",
33 sizeof(array));
34
35 // assegna dei valori agli elementi del VLA 1-D
36 for (size_t i = 0; i < arraySize; ++i) {
37 array[i] = i * i;
38 }
39
40 // assegna dei valori agli elementi del primo VLA 2-D
41 for (size_t i = 0; i < row1; ++i) {
42 for (size_t j = 0; j < col1; ++j) {
43 array2D1[i][j] = i + j;
44 }
45 }
46
47 // assegna dei valori agli elementi del secondo VLA 2-D
48 for (size_t i = 0; i < row2; ++i) {
49 for (size_t j = 0; j < col2; ++j) {
50 array2D2[i][j] = i + j;
51 }
52 }
53
54 puts("\nOne-dimensional array:");
55 print1DArray(arraySize, array); // VLA 1-D come argomento
56
57 puts("\nFirst two-dimensional array:");
58 print2DArray(row1, col1, array2D1); // VLA 2-D come argomento
59
60 puts("\nSecond two-dimensional array:");
61 print2DArray(row2, col2, array2D2); // VLA 2-D come argomento
62 }
63
64 void print1DArray(size_t size, int array[size]) {
65 // stampa i contenuti dell'array
66 for (size_t i = 0; i < size; i++) {
67 printf("array[%zu] = %d\n", i, array[i]);
68 }
69 }
70
71 void print2DArray(size_t row, size_t col, int array[row][col]) {
72 // stampa i contenuti dell'array
73 for (size_t i = 0; i < row; i++) {
74 for (size_t j = 0; j < col; j++) {
75 printf("%5d", array[i][j]);
76 }
77 }
78 }
```

```

77 puts("");
78 }
79 }
80 }

Enter size of a one-dimensional array: 6
Enter number of rows and columns in a 2-D array: 2 5
Enter number of rows and columns in another 2-D array: 4 3

sizeof(array) yields array size of 24 bytes

One-dimensional array:
array[0] = 0
array[1] = 1
array[2] = 4
array[3] = 9
array[4] = 16
array[5] = 25

First two-dimensional array:
0 1 2 3 4
1 2 3 4 5

Second two-dimensional array:
0 1 2
1 2 3
2 3 4
3 4 5

```

**Figura 6.18** Uso di array di lunghezza variabile in C99.

### Creare i VLA

Le righe 10-29 chiedono all'utente le dimensioni che desidera per un array unidimensionale e due array bidimensionali e usano i valori di input nelle righe 14, 21 e 29 per creare VLA. Queste righe sono valide purché le variabili che rappresentano le dimensioni degli array siano di un tipo intero.

### Operatore sizeof con VLA

Dopo aver creato gli array, usiamo l'operatore `sizeof` nelle righe 32-33 per controllare la lunghezza del nostro VLA unidimensionale. L'operatore `sizeof` è normalmente un'operazione della fase di compilazione, ma opera nella fase di esecuzione quando è applicato a un VLA. La finestra dell'output mostra che l'operatore `sizeof` restituisce una dimensione di 24 byte, quattro volte quella del numero che abbiamo inserito, perché la dimensione di un `int` sulla nostra macchina è di 4 byte.

### Assegnare valori a elementi di VLA

In seguito assegniamo valori agli elementi dei nostri VLA (righe 36-52). Usiamo la condizione di continuazione del ciclo `i < arraySize` quando riempiamo l'array unidimensionale. Come con gli array di lunghezza fissa, non vi è protezione contro gli accessi al di fuori dei confini dell'array.

### Funzione print1DArray

Le righe 64-69 definiscono la funzione `print1DArray` che stampa il suo argomento VLA unidimensionale. I parametri di funzione dei VLA hanno la stessa sintassi dei parametri dei normali array. Usiamo il parametro `size` nella dichiarazione del parametro `array`, ma è semplicemente una documentazione per il programmatore.

**Funzione print2DArray**

La funzione `print2DArray` (righe 71-80) stampa un VLA bidimensionale. Ricordate che dovete specificare una dimensione per tutto tranne che il primo indice di un parametro array multidimensionale. La stessa restrizione vale per i VLAs, eccetto che le dimensioni possono essere specificate da variabili. Il valore iniziale di `col` passato alla funzione determina dove ogni riga inizia nella memoria, proprio come con un array dalla dimensione fissa.

**✓ Autovalutazione**

1. (*Vero/Falso*) A differenza degli array di lunghezza fissa, i VLAs forniscono una protezione contro gli accessi al di fuori dei confini dell'array.

**Risposta:** *Falso*. In realtà, come con gli array di lunghezza fissa, non vi è protezione contro gli accessi al di fuori dei confini dell'array.

2. (*Vero/Falso*) `sizeof` è un'operazione della fase di compilazione.

**Risposta:** *Falso*. In realtà, normalmente `sizeof` è un'operazione della fase di compilazione, ma quando è applicata a un VLA, `sizeof` opera nella fase di esecuzione.



## 6.13 Programmazione sicura in C

**Controllo dei confini per gli indici di un array**

È importante assicurarsi che ogni indice che si usa per accedere a un elemento di un array sia dentro i confini dell'array. Gli indici di un array unidimensionale devono essere maggiori o uguali a 0 e minori del numero di elementi. Gli indici di riga e di colonna di un array bidimensionale devono essere maggiori o uguali a 0 e minori dei numeri, rispettivamente, delle righe e delle colonne. Questo si estende anche agli array di ulteriori dimensioni.

Permettere ai programmi di leggere o scrivere negli elementi di un array al di fuori dei confini degli array stessi è un comune difetto di sicurezza. Leggere da elementi di un array al di fuori dei confini può fare arrestare il programma o perfino dare l'impressione che questo venga eseguito correttamente, mentre invece usa dati non validi. Scrivere in elementi al di fuori dei confini (noto come overflow del buffer) può corrompere i dati in memoria di un programma, arrestare un programma e permettere agli autori di un attacco di sfruttare il sistema ed eseguire il proprio codice.

Il C non fornisce alcun controllo automatico dei confini per gli array. Spetta a voi assicurarvi che gli indici dell'array siano sempre maggiori o uguali a 0 e minori del numero di elementi dell'array. Per ulteriori tecniche utili a evitare tali problemi, fate riferimento alla linea guida CERT ARR30-C all'indirizzo <https://wiki.sei.cmu.edu/confluence>.

**scanf\_s**

Il controllo dei confini è pure importante nell'elaborazione di stringhe. Quando si legge una stringa ponendola in un array di `char`, `scanf` non previene automaticamente gli overflow del buffer. Se il numero di caratteri in input è maggiore o uguale alla lunghezza dell'array, `scanf` scriverà caratteri – compreso il carattere nullo di terminazione ('\0') – oltre la fine dell'array. Essa potrebbe quindi sovrascrivere altri valori di variabili in memoria. Inoltre, se il programma scrive in quelle posizioni il valore di altre variabili, potrebbe sovrascrivere il carattere '\0' della stringa.

Una funzione determina la terminazione delle stringhe cercando il loro carattere di terminazione '\0'. Per esempio, ricordatevi che la funzione `printf` stampa una stringa leggendo i caratteri dall'inizio della stringa in memoria fino a quando non incontra il carattere '\0' della stringa. Se il carattere '\0' manca, `printf` continua a leggere dalla memoria (e a stampare) finché non incontra qualche altro '\0' in memoria. Questo può portare a strani risultati o causare l'arresto di un programma.

L'Annex K opzionale del C11 standard fornisce nuove versioni più sicure di molte funzioni di elaborazione di stringhe e di input/output. Durante la lettura di una stringa in un array di caratteri la funzione `scanf_s` ese-

gue controlli per assicurarsi che non si scriva oltre la fine dell'array. Supponendo che `myString` sia un array di 20 caratteri, l'istruzione

```
scanf_s("%19s", myString, 20);
```

legge una stringa e la pone in `myString`. La funzione `scanf_s` richiede due argomenti per ogni `%s` nella stringa di formattazione:

- un array di caratteri in cui porre la stringa di input;
- il numero degli elementi dell'array.

La funzione usa il numero di elementi per evitare overflow del buffer. Per esempio, è possibile fornire un'ampiezza di campo per `%s` troppo lunga per l'array di caratteri sottostante o semplicemente omettere completamente l'ampiezza del campo. Con `scanf_s`, se il numero dei caratteri in input più il carattere di terminazione nullo è più grande del numero degli elementi dell'array specificato, la conversione `%s` fallisce. Per l'istruzione precedente, che contiene solo una specifica di conversione, `scanf_s` restituirebbe 0, indicando che non sono state eseguite conversioni. L'array `myString` rimarrebbe inalterato. Esamineremo le ulteriori funzioni dell'Annex K nei successivi paragrafi sulla programmazione sicura in C.

Non tutti i compilatori supportano le funzioni dell'Annex K del C11 standard. Per i programmi la cui compilazione deve avvenire su più piattaforme e compilatori può essere necessaria la modifica del codice per utilizzare le versioni di `scanf_s` o `scanf` disponibili su ciascuna piattaforma. Il vostro compilatore potrebbe anche richiedere una specifica impostazione per consentirvi l'utilizzo delle funzioni dell'Annex K.

### **Non usate stringhe inserite dall'utente come stringhe di controllo del formato**

Forse avete notato che in questo libro non usiamo istruzioni `printf` con un argomento singolo, e invece usiamo una delle seguenti forme.

- Quando dobbiamo stampare un '\n' dopo la stringa, usiamo la funzione `puts` (che automaticamente stampa un '\n' dopo la stringa singola che ha come argomento), come in

```
puts("Welcome to C!");
```

- Quando abbiamo necessità che il cursore rimanga sulla stessa riga della stringa, usiamo la funzione `printf`, come in

```
printf("%s", "Enter first integer: ");
```

Poiché in questi esempi stampiamo stringhe letterali, avremmo potuto certamente usare la forma di `printf`, come in

```
printf("Welcome to C!\n");
printf("Enter first integer: ");
```

Quando `printf` valuta la stringa di controllo del formato nel suo primo (ed eventualmente unico) argomento, esegue dei compiti in base alle specifiche di conversione in quella stringa. Se la stringa di controllo del formato fosse ottenuta dall'utente, un autore di attacchi potrebbe fornire specifiche di conversione dannose che verrebbero "eseguite" dalla funzione per l'output formattato. Ora che sapete come leggere le stringhe per porle in array di caratteri, è importante sottolineare che non dovete mai usare come stringa di controllo del formato di una `printf` un array di caratteri che potrebbe contenere degli input dell'utente. Per maggiori informazioni, fate riferimento alla linea guida CERT FIO30-C all'indirizzo <https://wiki.sei.cmu.edu/confluence>.

## ✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è falsa?

- a) È importante assicurarsi che ogni indice che si usa per accedere a un elemento di un array sia dentro i confini dell'array. Gli indici di un array unidimensionale devono essere maggiori o uguali a 0 e minori del numero di elementi dell'array. Gli indici di riga e di colonna di un array bidimensionale devono essere maggiori o uguali a 0 e minori dei numeri, rispettivamente, delle righe e delle colonne.

- b) Il C non fornisce alcun controllo automatico dei confini per gli array, così dovete preoccuparvene voi. Permettere ai programmi di leggere o scrivere negli elementi di un array al di fuori dei confini degli array stessi è un comune difetto di sicurezza.
- c) Leggere da elementi di un array al di fuori dei confini può fare arrestare il programma o perfino dare l'impressione che questo venga eseguito correttamente, mentre invece usa dati non validi.
- d) Tutte le affermazioni precedenti sono *vere*.

**Risposta:** d.

**2. (*Vero/Falso*)** Quando `printf` valuta la stringa di controllo del formato nel suo primo (ed eventualmente unico) argomento, la funzione esegue dei compiti in base alle specifiche di conversione in quella stringa. Se la stringa di controllo del formato fosse ottenuta dall'utente, un autore di attacchi potrebbe fornire specifiche di conversione dannose che verrebbero "eseguite" dalla funzione per l'output formattato. Non dovete mai usare come stringa di controllo del formato di una `printf` un array di caratteri che potrebbe contenere degli input dell'utente.

**Risposta:** *Vero*.

## 6.14 Riepilogo

### Paragrafo 6.1 Introduzione

- Gli array sono strutture di dati costituite da elementi di dati correlati dello stesso tipo.
- Gli array sono entità "statiche" in quanto rimangono della stessa dimensione per tutta l'esecuzione di un programma.

### Paragrafo 6.2 Array

- Un array è un **gruppo contiguo di locazioni di memoria** correlate dal fatto che tutte hanno lo stesso nome e lo stesso tipo.
- Per fare riferimento a una particolare **locazione** o a un particolare **elemento**, bisogna specificare il nome dell'array e il **numero di posizione** dell'elemento.
- Il primo elemento in ogni array è alla locazione 0.
- Il numero di posizione contenuto fra parentesi quadre è chiamato più formalmente **indice**. Un indice deve essere un intero o un'espressione intera.
- Le **parentesi** che racchiudono l'indice di un array sono un operatore con il livello più alto di precedenza.

### Paragrafo 6.3 Definire gli array

- Dovete specificare il tipo di ogni elemento e il **numero di elementi** nell'array così che il computer possa riservare la giusta quantità di memoria.
- Potete usare un **array di tipo char** per memorizzare una **stringa di caratteri**.

### Paragrafo 6.4 Esempi di array

- Il tipo **size\_t** rappresenta un **tipo intero senza segno**. Questo tipo è raccomandato per qualunque variabile che rappresenta la dimensione o gli indici di un array. Il file di intestazione `<stddef.h>` definisce `size_t`, spesso incluso in altri file di intestazione (come `<stdio.h>`).
- Gli elementi di un array possono essere inizializzati quando l'array è definito facendo seguire la definizione da un segno di uguale e da parentesi graffe, {}, contenenti una **lista di inizializzatori** separati da virgolette. Se vi sono meno inizializzatori degli elementi nell'array, i restanti elementi sono inizializzati a zero.
- L'istruzione `int n[10] = {0};` inizializza esplicitamente il primo elemento a zero e inizializza i restanti nove elementi a zero perché vi sono meno inizializzatori di quanti sono gli elementi nell'array.

- Se la dimensione di un array viene omessa in una definizione con una lista di inizializzatori, il compilatore determina il numero di elementi dell'array dal numero di inizializzatori.
- La **direttiva per il preprocessore #define** può definire una **costante simbolica**, ovvero un identificatore che il preprocessore sostituisce con un testo di sostituzione prima della compilazione del programma. Quando il programma è preprocessato, tutte le occorrenze della costante simbolica sono sostituite con il testo di sostituzione. L'uso di costanti simboliche per specificare le dimensioni di un array rende i programmi più leggibili e più **modificabili**.
- Il C **non effettua il controllo dei confini di un array** per evitare che un programma si riferisca a un elemento che non esiste. Così un programma che è in esecuzione può "andare oltre" la fine di un array senza messaggi di avvertimento. Dovete assicurarvi che tutti i riferimenti all'array rimangano entro i confini dell'array.

#### Paragrafo 6.5 Uso di array di caratteri per memorizzare e manipolare stringhe

- Una stringa letterale come "hello" è in realtà un array di caratteri individuali in C.
- È possibile inizializzare un **array di caratteri** usando una **stringa letterale**. In questo caso, la dimensione dell'array è determinata dal compilatore in base alla lunghezza della stringa.
- Ogni stringa contiene uno speciale **carattere di terminazione di stringa** chiamato **carattere nullo**. La costante di tipo carattere che rappresenta il carattere nullo è '\0'.
- Un array di caratteri che rappresenta una stringa deve sempre essere definito sufficientemente grande per contenere il numero dei caratteri nella stringa più il carattere nullo di terminazione.
- È possibile inizializzare gli array di caratteri anche con costanti individuali di tipo carattere in una lista di inizializzatori.
- Poiché una stringa è in realtà un array di caratteri, possiamo accedere ai suoi caratteri individuali usando direttamente la notazione con indice per gli array.
- Potete inserire una stringa direttamente nell'array di caratteri dalla tastiera usando `scanf` e la **specifica di conversione %s**. Il nome dell'array di caratteri è passato a `scanf` senza il carattere & che lo precede usato con variabili diverse dagli array.
- La funzione `scanf` legge i caratteri dalla tastiera fino a incontrare il primo carattere di spaziatura. Essa non effettua un controllo sulla dimensione dell'array. Così, `scanf` può scrivere oltre la fine dell'array. Per questo motivo, quando si legge una stringa in un array di caratteri con `scanf` si deve sempre usare un'ampiezza di campo minore di 1 rispetto alla dimensione dell'array di tipo `char` (per esempio, "%19s" per un array di 20 elementi).
- L'array di caratteri che rappresenta una stringa può essere stampato con `printf` e la specifica di conversione `%s`. I caratteri della stringa sono stampati fino a incontrare un carattere nullo di terminazione.

#### Paragrafo 6.6 Array locali statici e array locali automatici

- Possiamo applicare `static` alla definizione di un array locale così che l'array non sia creato e inizializzato a ogni chiamata della funzione e non sia distrutto ogni volta che si esce dalla funzione. Questo riduce il tempo di esecuzione del programma, particolarmente per i programmi con funzioni chiamate frequentemente che contengono array di grandi dimensioni.
- Gli array `static` sono automaticamente inizializzati una volta per tutte all'avvio del programma. Se non inizializzate esplicitamente un array `static`, i suoi elementi sono inizializzati a zero dal compilatore.

#### Paragrafo 6.7 Passare gli array alle funzioni

- Per passare un argomento array a una funzione, specificate il nome dell'array senza parentesi.
- Diversamente dagli array di tipo `char` che contengono stringhe, gli altri tipi di array non hanno un terminatore speciale. Per questa ragione, anche la dimensione di un array va passata a una funzione, così che la funzione possa processare il numero appropriato di elementi.

- Il C passa automaticamente gli array alle funzioni per riferimento: la funzione chiamata può modificare i valori degli elementi negli array originari della funzione chiamante. Il nome di un array ha come valore l'indirizzo del primo elemento dell'array. Poiché viene passato l'indirizzo di partenza dell'array, la funzione chiamata sa esattamente dove l'array è memorizzato e può modificare l'array originario nella funzione chiamante.
- Sebbene interi array vengano passati per riferimento, singoli elementi degli array sono passati per valore esattamente come semplici variabili.
- Singoli pezzi di dati (come singoli elementi di tipo `int`, `float` e `char`) sono chiamati **scalari**.
- Per passare un elemento di un array a una funzione, usate il nome con l'indice dell'elemento dell'array.
- Perché una funzione riceva un array tramite una chiamata di funzione, nella lista dei parametri della funzione si deve specificare che si dovrà ricevere un array. Non è necessaria la dimensione dell'array tra le parentesi dell'array. Se essa è inclusa, il compilatore controlla che sia maggiore di zero, poi la ignora.
- Se un parametro array è preceduto dal **qualificatore const**, ogni tentativo di modificare un elemento nel corpo della funzione produce un errore di compilazione.

#### Paragrafo 6.8 Ordinamento di array

- L'ordinamento di dati (ossia mettere i dati in ordine crescente o decrescente) è una delle più importanti applicazioni informatiche.
- Una tecnica di ordinamento è chiamata **bubble sort** o **sinking sort**, perché i valori più piccoli si assomilano a "bolle" che a poco a poco salgono verso la cima dell'array, come le bolle d'aria che risalgono nell'acqua, mentre i valori più grandi scendono verso il fondo dell'array. La tecnica consiste nell'effettuare diverse passate lungo l'array. A ogni passata vengono confrontate le successive coppie di elementi. Se una coppia è in ordine crescente (o se i valori sono identici), si lasciano i valori come sono. Se una coppia è in ordine decrescente, i loro valori sono scambiati nell'array.
- Tramite il bubble sort è possibile spostare un valore grande in giù lungo l'array di molte posizioni in una singola passata, mentre un valore piccolo può essere spostato in su solo di una posizione.
- Il pregio principale del bubble sort consiste nell'essere facile da programmare. Tuttavia, esso è molto lento e ciò risulta evidente quando si ordinano array di grandi dimensioni.

#### Paragrafo 6.9 Caso pratico di introduzione alla data science: analisi dei dati di un sondaggio

- La **media** è la media aritmetica di un insieme di valori.
- La **mediana** è il "valore centrale" in un insieme ordinato di valori.
- La **moda** è il valore che ricorre più frequentemente in un insieme di valori.

#### Paragrafo 6.10 Ricerca in array

- Il processo che consiste nel trovare un particolare elemento in un array è chiamato **ricerca**.
- La **ricerca lineare** confronta ogni elemento dell'array con una chiave di ricerca. Dal momento che l'array non è in un ordine particolare, è altrettanto probabile che il valore cercato venga trovato nel primo elemento quanto nell'ultimo. In media, pertanto, la chiave di ricerca sarà confrontata con metà degli elementi dell'array.
- Il metodo della ricerca lineare funziona bene per gli array piccoli o non ordinati. Per gli array ordinati si può usare l'algoritmo di ricerca binaria ad alta velocità.
- L'algoritmo di **ricerca binaria** elimina dalla ricerca una metà degli elementi in un array ordinato dopo ogni confronto. L'algoritmo localizza l'elemento centrale dell'array e lo confronta con la chiave di ricerca. Se sono uguali, la chiave di ricerca è stata trovata e viene restituito l'indice di quell'elemento. Se non sono uguali, il problema si riduce alla ricerca in una metà dell'array. Se la chiave di ricerca è minore dell'elemento centrale dell'array, viene effettuata la ricerca nella prima metà dell'array, altrimenti nella seconda. Se la chiave di ricerca non viene trovata nel sottoarray specificato, l'algoritmo viene ripetuto su

un quarto dell'array originario. La ricerca continua finché la chiave di ricerca non è uguale all'elemento centrale di un sottoarray, o finché il sottoarray non è costituito da un unico elemento che non è uguale alla chiave di ricerca (cioè la chiave di ricerca non è stata trovata).

- Nella ricerca binaria è possibile determinare il numero massimo di confronti richiesti per un array trovando la prima potenza di 2 maggiore del numero degli elementi dell'array.

### Paragrafo 6.11 Array multidimensionali

- Un uso comune degli **array multidimensionali** è quello di rappresentare **tabelle** di valori costituite da **righe e colonne**. Per identificare un particolare elemento della tabella, dobbiamo specificare due indici. Per convenzione, il primo identifica la riga dell'elemento e il secondo la sua colonna.
- Gli array che richiedono due indici per identificare un elemento particolare sono chiamati **array bidimensionali**. Gli array multidimensionali possono avere più di due indici.
- È possibile inizializzare un array multidimensionale al momento della definizione. I valori in una lista di inizializzatori di un array bidimensionale sono raggruppati per righe tra parentesi graffe. Se non vi è un numero di inizializzatori sufficiente per una data riga, i restanti elementi di quella riga sono inizializzati a 0.
- Il primo indice di una dichiarazione di parametro array multidimensionale non è necessario, mentre lo sono tutti quelli successivi. Il compilatore usa questi indici per determinare le locazioni in memoria degli elementi di array multidimensionali. Tutti gli elementi di un array sono memorizzati consecutivamente a prescindere dal numero degli indici. In un array bidimensionale, la prima riga viene memorizzata seguita dalla seconda e così via.
- Fornire i valori degli indici nella dichiarazione di parametro permette al compilatore di dire alla funzione come localizzare un elemento dell'array. In un array bidimensionale, ogni riga è fondamentalmente un array unidimensionale. Per localizzare un elemento in una particolare riga, il compilatore deve sapere quanti elementi vi sono in ogni riga in modo da poter saltare il numero appropriato di locazioni di memoria quando accede agli elementi di una determinata riga.

### Paragrafo 6.12 Array di lunghezza variabile

- Un **array di lunghezza variabile** è un array la cui dimensione è definita da un'espressione calcolata al momento dell'esecuzione.
- Quando è applicata a un array di lunghezza variabile, la funzione `sizeof` opera in fase di esecuzione.
- Gli array di lunghezza variabile in C sono opzionali: potrebbero non essere supportati dal vostro compilatore.

## Esercizi di autovalutazione

### 6.1 Completate ciascuna delle seguenti frasi.

- a) Le liste e le tabelle di valori sono memorizzate in \_\_\_\_\_.
- b) Il numero usato per riferirsi a un particolare elemento di un array è chiamato il suo \_\_\_\_\_.
- c) Una \_\_\_\_\_ va usata per specificare la dimensione di un array perché rende il programma più modificabile.
- d) Il processo di messa in ordine degli elementi di un array è chiamato \_\_\_\_\_ dell'array.
- e) Determinare se un array contiene un certo valore chiave si dice \_\_\_\_\_ nell'array.
- f) Un array che usa due indici è detto array \_\_\_\_\_.

### 6.2 Stabilite se le seguenti affermazioni sono *vere* o *falsa*. Se la risposta è *falsa*, spiegate il perché.

- a) Un array può memorizzare molti tipi differenti di valori.
- b) L'indice di un array può essere del tipo di dati `double`.
- c) Se in una lista di inizializzatori vi sono meno inizializzatori del numero degli elementi nell'array, i restanti elementi vengono inizializzati con l'ultimo valore della lista di inizializzatori.

- d) È un errore se una lista di inizializzatori contiene più inizializzatori di quanti sono gli elementi nell'array.
- e) L'elemento individuale di un array che viene passato a una funzione come un argomento della forma `a[i]` e viene modificato nella funzione chiamata conterrà poi il valore modificato nella funzione chiamante.
- 6.3 Osservate le istruzioni che seguono riguardanti un array chiamato `fractions`.
- Definite una costante simbolica `SIZE` con il testo di sostituzione 10.
  - Definite un array `double` con un numero uguale a `SIZE` di elementi e inizializzate gli elementi a 0.
  - Fate riferimento all'elemento 4 dell'array.
  - Assegnate il valore 1.667 al nono elemento dell'array.
  - Assegnate il valore 3.333 al settimo elemento dell'array.
  - Stampate gli elementi 6 e 9 dell'array con due cifre di precisione alla destra del punto decimale e mostrate l'output che viene stampato sullo schermo.
  - Stampate tutti gli elementi di un array usando un'istruzione di iterazione `for`. Usate la variabile `x` come variabile di controllo per il ciclo. Mostrate l'output.
- 6.4 Scrivete istruzioni per effettuare le seguenti operazioni.
- Definite `table` come un array intero con 3 righe e 3 colonne. Supponete che la costante simbolica `SIZE` sia stata definita come valore 3.
  - Quanti elementi contiene l'array `table`? Stampate il numero totale di elementi.
  - Usate un'istruzione di iterazione `for` per inizializzare ogni elemento di `table` alla somma dei suoi indici. Usate variabili `x` e `y` come variabili di controllo.
  - Stampate i valori di ogni elemento dell'array `table`. Supponete che l'array sia stato inizializzato con la definizione:
- ```
int table[SIZE][SIZE] = {{1, 8}, {2, 4, 6}, {5}};
```
- 6.5 Trovate l'errore in ognuno dei seguenti segmenti di programma e correggetelo.
- `#define SIZE 100;`
 - `SIZE = 10;`
 - `int b[10] = {0};`
`int i;`
`for (size_t i = 0; i <= 10; ++i) {`
 `b[i] = 1;`
`}`
 - `#include <stdio.h>;`
 - `int a[2][2] = {{1, 2}, {3, 4}};`
`a[1, 1] = 5;`
 - `#define VALUE = 120`

Risposte agli esercizi di autovalutazione

- 6.1 a) array. b) indice. c) costante simbolica. d) ordinamento. e) ricerca. f) bidimensionale.
- 6.2 a) *Falso*. Un array può memorizzare solo valori dello stesso tipo.
 b) *Falso*. L'indice di un array deve essere un intero o un'espressione intera.
 c) *Falso*. Il C inizializza automaticamente i restanti elementi a zero.
 d) *Vero*.
 e) *Falso*. Gli elementi individuali di un array sono passati per valore. Se invece l'intero array è passato a una funzione, le modifiche agli elementi saranno riflesse nell'array originario.
- 6.3 a) `#define SIZE 10`
 b) `double fractions[SIZE] = {0.0};`
 c) `fractions[4]`

d) `fractions[9] = 1.667;`
 e) `fractions[6] = 3.333;`
 f) `printf("%.2f %.2f\n", fractions[6], fractions[9]);`
Output: 3.33 1.67.
 g) `for (size_t x = 0; x < SIZE; ++x) {`
 `printf("fractions[%zu] = %f\n", x, fractions[x]);`
`}`

Output:

```

fractions[0] = 0.000000
fractions[1] = 0.000000
fractions[2] = 0.000000
fractions[3] = 0.000000
fractions[4] = 0.000000
fractions[5] = 0.000000
fractions[6] = 3.333000
fractions[7] = 0.000000
fractions[8] = 0.000000
fractions[9] = 1.667000

```

- 6.4 a) `int table[SIZE][SIZE];`
 b) Nove elementi. `printf("%d\n", SIZE * SIZE);`
 c) `for (size_t x = 0; x < SIZE; ++x) {`
 `for (size_t y = 0; y < SIZE; ++y) {`
 `table[x][y] = x + y;`
 `}`
`}`
 d) `for (size_t x = 0; x < SIZE; ++x) {`
 `for (size_t y = 0; y < SIZE; ++y) {`
 `printf("table[%d][%d] = %d\n", x, y, table[x][y]);`
 `}`
`}`

Output:

```

table[0][0] = 1
table[0][1] = 8
table[0][2] = 0
table[1][0] = 2
table[1][1] = 4
table[1][2] = 6
table[2][0] = 5
table[2][1] = 0
table[2][2] = 0

```

- 6.5 a) Errore: il punto e virgola alla fine della direttiva per il preprocessore `#define`.
 Correzione: eliminate il punto e virgola.
 b) Errore: assegnare un valore a una costante simbolica usando un'istruzione di assegnazione.
 Correzione: assegnate un valore alla costante simbolica in una direttiva per il preprocessore `#define` senza usare l'operatore di assegnazione, come in `#define SIZE 10`.
 c) Errore: fare riferimento a un elemento di un array al di fuori dei confini dell'array (`b[10]`).
 Correzione: cambiate il valore finale della variabile di controllo a 9 o cambiate `<=` in `<`.
 d) Errore: il punto e virgola alla fine della direttiva per il preprocessore `#include`.
 Correzione: eliminate il punto e virgola.
 e) Errore: indicizzazione dell'array fatta in modo scorretto.
 Correzione: cambiate l'istruzione in `a[1][1] = 5;`

- f) Errore: il valore di una costante simbolica non si definisce usando l'operatore `=`.
 Correzione: cambiate la direttiva per il preprocessore in `#define VALUE 120`.

Esercizi

- 6.6 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
- a) Il C memorizza liste di valori in _____.
 - b) Gli elementi di un array sono correlati dal fatto che _____.
 - c) Quando ci si riferisce all'elemento di un array, il numero della posizione contenuto entro le parentesi quadre è chiamato _____.
 - d) I nomi dei cinque elementi dell'array `p` sono _____, _____, _____, _____ e _____.
 - e) Il contenuto di un particolare elemento di un array è detto il _____ di quell'elemento.
 - f) Denominare un array, determinare il suo tipo e specificare il numero di elementi nell'array è detto _____ dell'array.
 - g) Il processo di porre gli elementi di un array nell'ordine crescente o in quello decrescente è chiamato _____.
 - h) In un array bidimensionale, il primo indice identifica la _____ di un elemento e il secondo la sua _____.
 - i) Un array m -per- n contiene _____ righe, _____ colonne e _____ elementi.
 - j) Il nome dell'elemento nella riga 3 e nella colonna 5 dell'array `d` è _____.
- 6.7 Stabilite quali delle seguenti affermazioni sono *vere* e quali *false*. Se *false*, spiegate il perché.
- a) Per fare riferimento a una particolare locazione o a un elemento in un array, specificate il nome dell'array e il valore del particolare elemento.
 - b) La definizione di un array riserva spazio per l'array.
 - c) Per indicare che per l'array intero `p` si devono riservare 100 locazioni, scrivete
`p[100];`
 - d) Un programma che inizializza a zero gli elementi di un array con 15 elementi deve contenere un'istruzione `for`.
 - e) Un programma che somma gli elementi di un array bidimensionale deve contenere istruzioni `for` annidate.
 - f) La media, la mediana e la moda del seguente insieme di valori sono rispettivamente 5, 6 e 7: 1, 2, 5, 6, 7, 7, 7.
- 6.8 Scrivete istruzioni per effettuare ognuna delle seguenti operazioni.
- a) Stampare il valore del settimo elemento di un array di caratteri `f`.
 - b) Inserire un valore nell'elemento 4 di un array `b` unidimensionale di elementi in virgola mobile.
 - c) Inizializzare a 8 ognuno dei cinque elementi di un array intero unidimensionale `g`.
 - d) Sommare gli elementi dell'array `c` di 100 elementi in virgola mobile.
 - e) Copiare l'array `a` nella prima porzione dell'array `b`. Supponete che `a` abbia 11 elementi, `b` ne abbia 34 ed entrambi abbiano lo stesso tipo di elementi.
 - f) Determinare e stampare il valore più piccolo e il valore più grande contenuti nell'array `w` di 99 elementi in virgola mobile.
- 6.9 Considerate un array intero `t` di dimensioni 2-per-5.
- a) Scrivete una definizione per `t`.
 - b) Quante righe ha `t`?
 - c) Quante colonne ha `t`?
 - d) Quanti elementi ha `t`?
 - e) Scrivete i nomi di tutti gli elementi nella seconda riga di `t`.
 - f) Scrivete i nomi di tutti gli elementi nella terza colonna di `t`.
 - g) Scrivete un'istruzione singola che imposti a zero l'elemento di `t` nella riga 1 e nella colonna 2.

- h) Scrivete una serie di istruzioni che inizializzino a zero ogni elemento di `t`. Non usate un'istruzione di iterazione.
- i) Scrivete un'istruzione `for` annidata che inizializzi a zero ogni elemento di `t`.
- j) Scrivete un'istruzione che faccia inserire i valori per gli elementi di `t` dal terminale.
- k) Scrivete una serie di istruzioni che determinino e stampino il valore più piccolo nell'array `t`.
- l) Scrivete un'istruzione che stampi gli elementi della prima riga di `t`.
- m) Scrivete un'istruzione che sommi gli elementi della quarta colonna di `t`.
- n) Scrivete una serie di istruzioni che stampino l'array `t` in formato tabellare. Elencate gli indici di colonna come intestazioni al di sopra delle rispettive colonne ed elencate gli indici di riga alla sinistra delle rispettive righe.
- 6.10 (Commissioni sulle vendite)** Usate un array unidimensionale per risolvere il seguente problema. Un'azienda paga i suoi agenti di vendita su commissione. Gli agenti di vendita ricevono \$200 alla settimana più il 9% delle loro vendite lorde per quella settimana. Per esempio, un agente di vendita che ottiene un introito lorde sulle vendite di \$3000 in una settimana riceve \$200 più il 9% di \$3000, ovvero un totale di \$470. Scrivete un programma in C (usando un array di contatori) che determini quanti agenti di vendita hanno avuto i loro guadagni in ognuno dei seguenti intervalli (supponete che il guadagno di ogni agente di vendita sia troncato a una quantità intera):
- \$200-299
 - \$300-399
 - \$400-499
 - \$500-599
 - \$600-699
 - \$700-799
 - \$800-899
 - \$900-999
 - \$1000 e oltre
- 6.11 (Bubble sort)** Il bubble sort presentato nella Figura 6.12 è inefficiente per array grandi. Apportate le seguenti modifiche per migliorare le sue prestazioni:
- Dopo la prima passata, è garantito che il numero più grande si trova nell'elemento dell'array con il numero più alto; dopo la seconda passata, i due numeri più grandi sono "al loro posto", e così via. Invece di fare nove confronti a ogni passata, modificate il bubble sort per fare otto confronti alla seconda passata, sette alla terza passata e così via.
 - I dati nell'array possono già essere nell'ordine giusto o quasi giusto, allora perché fare nove passate se ne basterebbero di meno? Modificate l'ordinamento per controllare alla fine di ogni passata se sono stati fatti degli scambi. Se non ne sono stati fatti, allora i dati devono già essere nell'ordine giusto, così l'ordinamento dovrebbe terminare. Se sono stati fatti degli scambi, allora è necessaria almeno un'altra passata.
- 6.12** Scrivete dei cicli che effettuino ognuna delle seguenti operazioni su array unidimensionali:
- Inizializzare a zero i 10 elementi dell'array intero `counts`.
 - Aggiungere 1 a ognuno dei 15 elementi dell'array intero `bonus`.
 - Leggere i 12 valori dell'array `monthlyTemperatures` di tipo `float` dalla tastiera.
 - Stampare i cinque valori dell'array intero `bestScores` nel formato a colonne.
- 6.13** Trovate l'errore o gli errori in ognuna delle seguenti istruzioni:
- Presupponete: `char str[5] = "";`
`scanf("%s", str); // L'utente scrive hello`
 - Presupponete: `int a[3];`
`printf("$d %d %d\n", a[1], a[2], a[3]);`
 - `double f[3] = {1.1, 10.01, 100.001, 1000.0001};`
 - Presupponete: `double d[2][10] = {0};`
`d[1, 9] = 2.345;`

6.14 (Modifiche al programma per media, mediana e moda) Modificate il programma della Figura 6.13 in modo che la funzione mode possa trattare una situazione di parità per il valore della moda. Se ci sono due valori che ricorrono con la medesima frequenza, i dati sono “bimodali” e devono essere stampati entrambi i valori. Se ci sono più di due valori che ricorrono con la medesima frequenza, i dati sono “multimodali” e devono essere stampati tutti i valori con la stessa frequenza. Modificate anche la funzione median in modo che venga calcolata la media dei due elementi centrali in un array con un numero pari di elementi.

6.15 (Eliminazione di duplicati) Usate un array unidimensionale per risolvere il seguente problema. Memorizzate 20 numeri, ognuno dei quali compreso tra 10 e 100, estremi inclusi. Quando un numero viene letto, stampatelo solo se non è un duplicato di un numero già letto. Tenete conto del “caso peggiore”, in cui tutti i 20 numeri sono differenti. Usate l’array più piccolo possibile per risolvere questo problema.

6.16 Etichettate gli elementi dell’array bidimensionale sales 3-per-5 per indicare l’ordine in cui sono posti a zero dal seguente segmento di programma:

```
for (size_t row = 0; row <= 2; ++row) {
    for (size_t column = 0; column <= 4; ++column) {
        sales[row][column] = 0;
    }
}
```

6.17 Cosa fa il seguente programma?

```
1 // ex06_17.c
2 // Cosa fa questo programma?
3 #include <stdio.h>
4 #define SIZE 10
5
6 int whatIsThis(const int b[], size_t p); // prototipo di funzione
7
8 int main(void) {
9     // inizializza l'array a
10    int a[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
11
12    int x = whatIsThis(a, SIZE);
13    printf("Result is %d\n", x);
14 }
15
16 // cosa fa questa funzione?
17 int whatIsThis(const int b[], size_t p) {
18     if (1 == p) { // caso di base
19         return b[0];
20     }
21     else { // passo di ricorsione
22         return b[p - 1] + whatIsThis(b, p - 1);
23     }
24 }
```

6.18 Cosa fa il seguente programma?

```
1 // ex06_18.c
2 // Cosa fa questo programma?
3 #include <stdio.h>
4 #define SIZE 10
5
6 // prototipo di funzione
7 void someFunction(const int b[], size_t start, size_t size);
```

```

8
9 // la funzione main inizia l'esecuzione del programma
10 int main(void) {
11     int a[SIZE] = {8, 3, 1, 2, 6, 0, 9, 7, 4, 5}; // inizializza a
12
13     puts("Answer is:");
14     someFunction(a, 0, SIZE);
15     puts("");
16 }
17
18 // Cosa fa questa funzione?
19 void someFunction(const int b[], size_t start, size_t size) {
20     if (start < size) {
21         someFunction(b, start + 1, size);
22         printf("%d ", b[start]);
23     }
24 }
```

6.19 (Lancio dei dadi) Scrivete un programma che simuli il lancio di due dadi. Il programma deve usare due volte `rand` per lanciare, rispettivamente, il primo dado e il secondo dado, poi deve calcolare la somma dei due valori. Poiché ogni dado può mostrare sulla faccia superiore un valore intero da 1 a 6, la somma dei due valori varierà allora da 2 a 12, con 7 che è la somma più frequente e 2 e 12 che sono le somme meno frequenti. Il diagramma seguente mostra 36 possibili combinazioni dei due dadi.

1	2	3	4	5	6	
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Il vostro programma deve lanciare i due dadi 36.000 volte. Usate un array unidimensionale per annotare il numero delle volte in cui compare ogni possibile somma. Stampate i risultati in un formato tabellare. Stabilite inoltre se i totali sono ragionevoli: per esempio, vi sono sei modi di ottenere un risultato 7, quindi approssimativamente un sesto di tutti i lanci deve avere come somma 7.

6.20 (Statistiche del gioco Craps) Scrivete un programma che esegua 1.000.000 di volte il gioco Craps (senza alcun intervento umano) e risponda a ognuna delle seguenti domande.

- Quanti giochi vengono vinti al primo lancio, al secondo lancio, ..., al ventesimo lancio e oltre?
- Quanti giochi vengono persi al primo lancio, al secondo lancio, ..., al ventesimo lancio e oltre?
- Quali sono le possibilità di vincere a Craps? Dovreste scoprire che il Craps è uno dei giochi da casinò più corretti. Cosa pensate che ciò significhi?
- Qual è la lunghezza media di una partita di Craps?
- Le possibilità di vincere aumentano con la durata del gioco?

6.21 (Sistema di prenotazione per compagnie aeree) Una piccola compagnia aerea ha appena acquistato un computer per il suo nuovo sistema automatico di prenotazione. Il presidente vi ha chiesto di programmare il nuovo sistema. Scrivrete un programma per assegnare i posti su ogni volo dell'unico aereo della compagnia (capacità: 10 posti). Il vostro programma deve stampare il seguente menu di alternative:

```

Please type 1 for "first class"
Please type 2 for "economy"
```

Se la persona scrive 1, allora il vostro programma deve assegnare un posto in prima classe (posti da 1 a 5). Se la persona scrive 2, allora il vostro programma deve assegnare un posto in classe economica (posti da 6 a 10). Il vostro programma deve quindi stampare una carta d'imbarco indicante il numero del posto della persona e se questo si trova in prima classe o in classe economica.

Usate un array unidimensionale per rappresentare la mappa dei posti dell'aereo. Inizializzate tutti gli elementi dell'array a 0 per indicare che tutti i posti sono vuoti. Quando ogni posto viene assegnato, impostate a 1 l'elemento corrispondente dell'array per indicare che il posto non è più disponibile.

Il vostro programma, naturalmente, non deve mai assegnare un posto che è già stato assegnato. Quando la prima classe è piena, il vostro programma deve domandare alla persona se è disposta ad accettare un posto in classe economica (e viceversa). Se lo è, assegnate il posto appropriato; se non lo è, stampate il messaggio, "Next flight leaves in 3 hours".

6.22 (Totale delle vendite) Usate un array bidimensionale per risolvere il seguente problema. Un'azienda ha quattro agenti di vendita (da 1 a 4) che vendono cinque differenti prodotti (da 1 a 5). Una volta al giorno, ogni agente di vendita consegna una distinta per ogni tipo differente di prodotto venduto. Ogni distinta contiene:

- il numero dell'agente di vendita;
- il numero del prodotto;
- il valore totale in dollari di quel prodotto venduto quel giorno.

In questo modo, ogni agente di vendita consegna tra 0 e 5 distinte di vendita al giorno. Supponete che siano disponibili le informazioni contenute in tutte le distinte per l'ultimo mese. Scrivete un programma che legga queste informazioni per le vendite dell'ultimo mese e riepiloghi le vendite totali per agente di vendita e per prodotto. Tutti i totali devono essere memorizzati nell'array bidimensionale `sales`. Dopo aver elaborato tutte le informazioni per l'ultimo mese, stampate i risultati in formato tabellare con ogni colonna che rappresenta uno specifico agente di vendita e ogni riga che rappresenta uno specifico prodotto. Calcolate il totale parziale di ogni riga per ottenere il totale delle vendite di ogni prodotto per l'ultimo mese; calcolate il totale parziale di ogni colonna per ottenere il totale delle vendite realizzato da ogni agente di vendita per l'ultimo mese. La vostra stampa tabellare deve includere questi totali parziali alla destra delle righe sommate e in fondo alle colonne sommate.

6.23 (Grafici a tartaruga) Il linguaggio Logo ha reso famoso il concetto dei *grafici a tartaruga*. Immaginate una tartaruga meccanica che cammini intorno in una stanza sotto il controllo di un programma in C. La tartaruga tiene una penna che può trovarsi in una di due posizioni, in su o in giù. Finché la penna sta in giù, la tartaruga traccia delle forme mentre si muove; finché la penna sta in su, la tartaruga si muove intorno liberamente senza scrivere nulla. In questo problema simulerete l'operazione della tartaruga e creerete anche un foglio da disegno computerizzato.

Usate un array `floor` 50-per-50 inizializzato a zero. Leggete i comandi da un array che li contiene. Tenete continuamente traccia della posizione corrente della tartaruga e della posizione della penna, in su o in giù. Supponete che la tartaruga parta sempre alla posizione 0, 0 del pavimento con la sua penna in su. L'insieme di comandi della tartaruga che il vostro programma deve elaborare è mostrato nella tabella seguente:

Comando	Significato
1	Penna in su
2	Penna in giù
3	Gira a destra
4	Gira a sinistra
5, 10	Spostati in avanti di 10 posizioni (o di un numero diverso da 10)
6	Stampa l'array 50-per-50
9	Fine dei dati (sentinella)

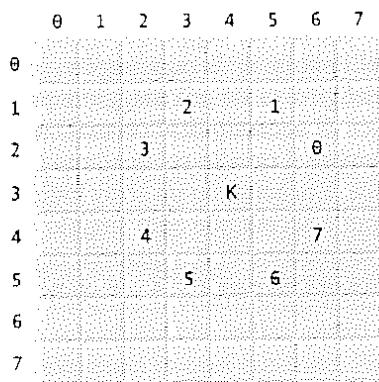
Supponete che la tartaruga sia da qualche parte vicino al centro del pavimento. Il seguente "programma" disegna e stampa un quadrato di 12-per-12:

```
2
5,12
3
5,12
3
5,12
3
5,12
1
6
9
```

Quando la tartaruga si muove con la penna in giù, impostate a 1 gli elementi dell'array `floor`. Quando viene dato il comando 6, stampate un asterisco ovunque vi sia un 1 nell'array. Per ogni zero, stampate uno spazio. Scrivete un programma per implementare le funzionalità dei grafici a tartaruga esaminate qui. Scrivete diversi programmi per grafici a tartaruga per disegnare forme interessanti. Aggiungete altri comandi per aumentare la potenza del vostro linguaggio per i grafici a tartaruga.

6.24 (*Il Giro del Cavallo*) Uno dei puzzle più interessanti per gli appassionati di scacchi è il problema del Giro del Cavallo, originariamente proposto dal matematico Eulero. Il problema è questo: può il pezzo degli scacchi chiamato "cavalo" muoversi su una scacchiera vuota e toccare ognuno dei 64 quadrati una volta e una volta soltanto? Studiamo qui in maniera approfondita questo intrigante problema.

Il cavallo compie movimenti a forma di L (facendo mosse di due quadrati in una direzione e poi di un quadrato in una direzione perpendicolare). In questo modo, da un quadrato al centro di una scacchiera vuota, il cavallo può compiere otto differenti mosse (numerate da 0 a 7) come mostrato nel diagramma seguente:



- Disegnate su un foglio di carta una scacchiera 8-per-8 e tentate a mano un giro del cavallo. Mettete un 1 nel primo quadrato da cui vi muovete, un 2 nel secondo quadrato, un 3 nel terzo, e così via. Prima di iniziare il giro, fate una stima di fin dove pensate di arrivare, ricordando che un giro completo consta di 64 mosse. Fin dove siete arrivati? Vi siete avvicinati alla vostra stima?
- Ora sviluppiamo un programma che muova il cavallo su una scacchiera. La scacchiera stessa è rappresentata da un array bidimensionale `board` 8-per-8. Ogni quadrato è inizializzato a zero. Descriviamo ognuna delle otto possibili mosse in termini delle sue componenti sia orizzontali che verticali. Per esempio, una mossa di tipo 0, come mostrato nel precedente diagramma, consiste nel muoversi di due quadrati orizzontalmente verso destra e di un quadrato verticalmente verso l'alto. La mossa 2 consiste nel muoversi di un quadrato orizzontalmente verso sinistra e due quadrati verticalmente verso l'alto. I movimenti orizzontali verso sinistra e i movimenti verticali verso l'alto sono indicati

con numeri negativi. È possibile descrivere le otto mosse con due array con singolo indice, `horizontal` e `vertical`, come segue:

```
horizontal[0] = 2      vertical[0] = -1
horizontal[1] = 1      vertical[1] = -2
horizontal[2] = -1     vertical[2] = -2
horizontal[3] = -2     vertical[3] = -1
horizontal[4] = -2     vertical[4] = 1
horizontal[5] = -1     vertical[5] = 2
horizontal[6] = 1      vertical[6] = 2
horizontal[7] = 2      vertical[7] = 1
```

Le variabili `currentRow` e `currentColumn` indicano la riga e la colonna della posizione corrente del cavallo. Per compiere una mossa di tipo `moveNumber`, dove `moveNumber` è un valore compreso tra 0 e 7, il vostro programma userà le istruzioni

```
currentRow += vertical[moveNumber];
currentColumn += horizontal[moveNumber];
```

Definite un contatore che conti da 1 a 64. Registrate il valore corrente del contatore in ogni quadrato visitato dal cavallo. Ricordatevi di verificare ogni mossa potenziale per vedere se il cavallo ha già visitato quel quadrato e, naturalmente, verificate ogni mossa potenziale per assicurarvi che il cavallo non finisca fuori dalla scacchiera. Adesso scrivete un programma per muovere il cavallo sulla scacchiera. Eseguite il programma. Quante mosse ha fatto il cavallo?

- c) Dopo aver tentato di scrivere e far eseguire un programma del Giro del Cavallo, avete probabilmente avuto alcune preziose intuizioni. Le useremo per costruire un'*euristica* (o strategia) per muovere il cavallo. Le euristiche non garantiscono il successo, ma un'euristica sviluppata con attenzione ne aumenta molto la possibilità. Forse avete osservato che i quadrati esterni sono in un certo senso più fastidiosi dei quadrati più vicini al centro della scacchiera. In realtà, i quadrati più fastidiosi, o inaccessibili, sono i quattro angoli.

L'intuizione può suggerire che dovete dapprima tentare di muovere il cavallo verso i quadrati più fastidiosi e lasciare liberi quelli più facili da raggiungere, in modo che, quando la scacchiera comincia a riempirsi verso la fine del giro, ci sia una maggiore possibilità di successo.

Sviluppiamo una "euristica di accessibilità" classificando ogni quadrato in base a quanto esso sia accessibile e spostando sempre il cavallo sul quadrato (fra i movimenti a forma di L del cavallo, naturalmente) più inaccessibile. Etichettiamo un array bidimensionale `accessibility` con numeri che indicano da quanti quadrati è accessibile ogni singolo quadrato. Su una scacchiera vuota i quadrati centrali ottengono una valutazione di 8, i quadrati degli angoli una valutazione di 2, e gli altri quadrati hanno numeri di accessibilità pari a 3, 4 o 6 come segue:

2	3	4	4	4	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

Adesso scrivete una versione del programma del Giro del Cavallo usando l'euristica di accessibilità. A ogni passo il cavallo deve spostarsi sul quadrato con il numero di accessibilità più basso. In caso di valori equivalenti, il cavallo può spostarsi su uno qualsiasi dei quadrati con lo stesso valore. Il giro può pertanto iniziare in uno qualsiasi dei quattro angoli. [Nota: mentre il cavallo si muove sulla scacchiera, il vostro programma deve ridurre i numeri di accessibilità man mano che vengono occupati sempre più quadrati. In questo modo, a un dato momento durante il giro, il numero di accessibilità per ogni quadrato disponibile sarà esattamente uguale al numero di quadrati da cui si può raggiungere]

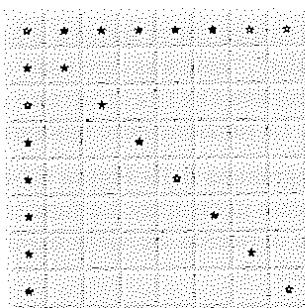
re quel quadrato.] Eseguite questa versione del vostro programma. Siete riusciti a compiere un giro completo? (*Facoltativo*: modificate il programma per eseguire 64 giri, uno da ogni quadrato della scacchiera. Quanti giri completi siete riusciti a fare?)

- d) Scrivete una versione del programma del Giro del Cavallo che, quando incontra lo stesso valore per due o più quadrati, decida quale quadrato scegliere guardando in avanti i valori di quei quadrati raggiungibili dai quadrati con lo stesso valore. Il vostro programma deve muovere il cavallo sul quadrato per il quale il movimento successivo porta a un quadrato con il numero di accessibilità più basso.

6.25 (Giro del Cavallo: approcci a forza bruta) Nell'Esercizio 6.24 abbiamo sviluppato una soluzione per il problema del Giro del Cavallo. L'approccio usato, chiamato "euristica di accessibilità", genera molte soluzioni e opera in maniera efficiente. Man mano che i computer diventano più efficienti, potremo risolvere molti problemi grazie alla loro potenza e con algoritmi relativamente poco sofisticati. Chiamiamo questo approccio alla risoluzione di problemi "a forza bruta".

- Usate i numeri casuali per permettere al cavallo di muoversi sulla scacchiera a caso nei suoi movimenti permessi a forma di L. Il vostro programma deve eseguire un giro e stampare la scacchiera finale. Fino a dove arriva il cavallo?
- Molto probabilmente, il precedente programma produce un giro relativamente breve. Adesso modificate il vostro programma per provare 1.000 giri. Usate un array unidimensionale per tenere traccia del numero di giri per ogni lunghezza. Quando il vostro programma finisce di provare i 1.000 giri, deve stampare queste informazioni in un formato tabellare. Qual è stato il migliore risultato?
- Molto probabilmente, il precedente programma vi ha dato alcuni giri "rispettabili" ma non giri completi. Ora "eliminate tutti gli stop" e lasciate semplicemente eseguire il programma finché non produce un ciclo completo. [*Precavzione*: questa versione del programma potrebbe richiedere ore su un computer potente.] Di nuovo, tenete una tabella per registrare il numero di giri di ogni lunghezza e stampatela quando viene trovato il primo giro completo. Quanti giri ha dovuto provare il vostro programma prima di produrre un giro completo? Quanto tempo ci è voluto?
- Confrontate la versione a forza bruta del Giro del Cavallo con la versione che usa l'euristica di accessibilità. Quale richiede uno studio più attento del problema? Quale algoritmo è stato più difficile da sviluppare? Quale ha richiesto più potenza del computer? Potremmo essere certi (in anticipo) di ottenere un giro completo con l'approccio euristico di accessibilità? Potremmo essere certi (in anticipo) di ottenere un giro completo con l'approccio a forza bruta? Discutete i pro e i contro della risoluzione dei problemi con la forza bruta in generale.

6.26 (Otto Regine) Un altro puzzle per gli appassionati di scacchi è il problema delle Otto Regine. Viene enunciato come segue: è possibile mettere otto regine su una scacchiera vuota in modo che nessuna regina ne "attacchi" un'altra, cioè in modo che due regine non stiano sulla stessa riga, sulla stessa colonna o lungo la stessa diagonale? Usate il genere di approccio sviluppato nell'Esercizio 6.24 per formulare un'euristica per risolvere il problema delle Otto Regine. Eseguite il vostro programma. *Suggerimento*: è possibile assegnare un valore numerico a ogni quadrato della scacchiera che indica quanti quadrati di una scacchiera vuota vanno "eliminati" una volta che una regina è posizionata in quel quadrato. Per esempio, a ognuno dei quattro angoli sarebbe assegnato il valore 22, come mostrato nel seguente diagramma:



Una volta che questi “numeri di eliminazione” sono assegnati a tutti i 64 quadrati, un approccio euristico potrebbe essere: posizionate la successiva regina nel quadrato con il numero di eliminazione più piccolo. Perché questa strategia è intuitivamente attraente?

6.27 (*Otto Regine: approcci a forza bruta*) In questo esercizio svilupperete diversi approcci a forza bruta per risolvere il problema delle Otto Regine introdotto nell’Esercizio 6.26.

- Risolvete il problema delle Otto Regine usando la tecnica casuale a forza bruta sviluppata nell’Esercizio 6.25.
- Usate una tecnica esaustiva (ossia, tentate tutte le possibili combinazioni delle otto regine sulla scacchiera).
- Perché ritenete che l’approccio esaustivo a forza bruta non possa essere appropriato per risolvere il problema delle Otto Regine?
- Mettete a confronto gli approcci a forza bruta casuali e gli approcci a forza bruta esaustivi in generale.

6.28 (*Eliminazione di duplicati*) Nel Capitolo 12 esploreremo la struttura di dati ad albero per la ricerca binaria ad alta velocità. Una caratteristica dell’albero di ricerca binaria è quella che i valori duplicati sono scartati quando vengono fatte inserzioni al suo interno. Questa è detta eliminazione dei duplicati. Scrivete un programma che produca 20 numeri a caso tra 1 e 20. Il programma deve memorizzare tutti i valori non duplicati in un array. Usate l’array più piccolo possibile per eseguire questo compito.

6.29 (*Giro del Cavallo: test di chiusura del giro*) Nel Giro del Cavallo si ha un giro completo quando il cavallo compie 64 mosse toccando ogni quadrato della scacchiera una volta e una volta soltanto. Si ha un giro chiuso quando la 64^{ma} mossa colloca il cavallo in una posizione lontana una mossa da quella da cui ha iniziato il giro. Modificate il programma del Giro del Cavallo che avete scritto nell’Esercizio 6.24 per verificare se si è in presenza di un giro chiuso quando si ha un giro completo.

6.30 (*Il Setaccio di Eratostene*) Un numero primo è un intero maggiore di 1 divisibile solo per se stesso e per 1. In questo esercizio, userete il Setaccio di Eratostene per trovare tutti i numeri primi inferiori a 1.000. Esso opera come segue.

- Create un array di 100 elementi, tutti inizializzati a 1 (vero). Gli elementi dell’array con indici primi rimarranno con valore 1. Tutti gli altri elementi dell’array saranno alla fine impostati a zero.
- Partendo dall’indice 2 (l’indice 1 non è primo), ogni volta che si trova un elemento dell’array il cui valore è 1, effettuate un’iterazione lungo il resto dell’array e impostate a zero ogni elemento il cui indice è un multiplo dell’indice dell’elemento con valore 1. Per l’indice 2 dell’array, tutti gli elementi che seguono nell’array che sono multipli di 2 saranno impostati a zero (quelli con gli indici 4, 6, 8, 10 e così via.). Per l’indice 3 dell’array, tutti gli elementi successivi nell’array che sono multipli di 3 saranno impostati a zero (quelli con indici 6, 9, 12, 15 e così via).

Al termine di questo processo, gli elementi dell’array che hanno ancora il valore 1 indicano che l’indice corrispondente è un numero primo. Scrivete un programma che determini e stampi i numeri primi tra 1 e 999. Ignorate l’elemento 0 dell’array.

Esercizi sulla ricorsione

6.31 (*Palindromi*) Un palindromo è una stringa che si scrive e si legge allo stesso modo in avanti e all’indietro. Alcuni esempi di palindromi sono: “radar”, “able was i ere i saw elba” e, se ignorate gli spazi, “a man a plan a canal panama”. Scrivete una funzione ricorsiva `testPalindrome` che restituisca 1 se la stringa memorizzata in un array è un palindromo e altrimenti 0. La funzione deve ignorare gli spazi e la punteggiatura nella stringa.

6.32 (*Ricerca lineare*) Modificate il programma della Figura 6.14 in modo da usare una funzione ricorsiva `linearSearch` per eseguire la ricerca lineare in un array. La funzione deve ricevere come argomenti un array intero, la dimensione dell’array e la chiave di ricerca. Se viene trovata la chiave di ricerca, essa deve restituire l’indice corrispondente dell’array, altrimenti -1.

6.33 (Ricerca binaria) Modificate il programma della Figura 6.15 in modo da usare una funzione ricorsiva `binarySearch` per eseguire la ricerca binaria in un array. La funzione deve ricevere come argomenti un array intero, l'indice iniziale, l'indice finale e la chiave di ricerca. Se viene trovata la chiave di ricerca, essa deve restituire l'indice dell'array, altrimenti `-1`.

6.34 (Otto Regine) Modificate il programma delle Otto Regine che avete sviluppato nell'Esercizio 6.26 per risolvere il problema in maniera ricorsiva.

6.35 (Stampare un array) Scrivete una funzione ricorsiva `printArray` che riceva come argomenti un array e la dimensione dell'array, stampi l'array e non restituisca niente. La funzione deve arrestare l'elaborazione e tornare alla funzione chiamante quando essa riceve un array di dimensione zero.

6.36 (Stampare una stringa all'indietro) Scrivete una funzione ricorsiva `stringReverse` che riceva un array di caratteri come argomento, lo stampi all'indietro e non restituisca niente. La funzione deve arrestare l'elaborazione e tornare alla funzione chiamante quando incontra il carattere nullo di terminazione della stringa.

6.37 (Trovare il valore minimo in un array) Scrivete una funzione ricorsiva `recursiveMinimum` che riceva come argomenti un array intero e la dimensione dell'array e restituisca l'elemento più piccolo dell'array. La funzione deve arrestare l'elaborazione e tornare alla funzione chiamante quando riceve un array di un solo elemento.

CAPITOLO

7

Sommario del capitolo

- 7.1 Introduzione
- 7.2 Definizioni e inizializzazione di variabili puntatore
- 7.3 Operatori per i puntatori
- 7.4 Passare argomenti a funzioni per riferimento
- 7.5 Uso del qualificatore `const` con i puntatori
- 7.6 Bubble sort che utilizza il passaggio per riferimento
- 7.7 Operatore `sizeof`
- 7.8 Espressioni con puntatori e aritmetica dei puntatori
- 7.9 Relazioni tra puntatori e array
- 7.10 Array di puntatori
- 7.11 Caso pratico di simulazione di numeri casuali: mescolare e distribuire le carte
- 7.12 Puntatori a funzioni
- 7.13 Programmazione sicura in C
- 7.14 Riepilogo

Puntatori

Obiettivi

- Usare i puntatori e gli operatori per i puntatori.
- Passare argomenti alle funzioni per riferimento usando i puntatori.
- Comprendere le varie collocazioni del qualificatore `const` e il modo in cui influiscono su ciò che è possibile fare con una variabile.
- Usare l'operatore `sizeof` con variabili e tipi.
- Usare l'aritmetica dei puntatori per elaborare gli elementi negli array.
- Comprendere le strette relazioni tra puntatori, array e stringhe.
- Definire e usare array di stringhe.
- Usare i puntatori a funzioni.
- Approfondire le questioni riguardanti la programmazione sicura in C relativamente ai puntatori.

7.1 Introduzione

In questo capitolo analizzeremo uno dei costrutti più potenti del linguaggio di programmazione C, il **puntatore**. I puntatori permettono ai programmi di

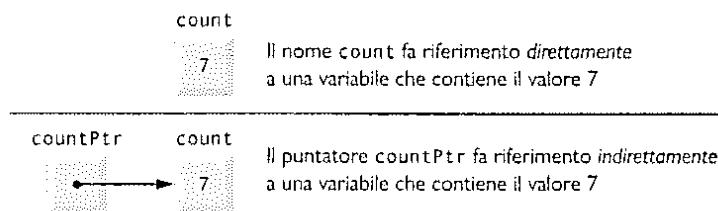
- realizzare il passaggio per riferimento;
- passare le funzioni a funzioni;
- manipolare stringhe e array;
- creare e manipolare strutture dinamiche di dati, ossia quelle che possono crescere e contrarsi al momento dell'esecuzione, come liste collegate, code, pile e alberi.

Questo capitolo spiegherà i concetti fondamentali sui puntatori. Nel Paragrafo 7.13 discuteremo varie questioni di sicurezza legate ai puntatori. Il Capitolo 10 esaminerà l'utilizzo di puntatori con le strutture. Il Capitolo 12 introdurrà la gestione dinamica della memoria e mostrerà come creare e usare strutture di dati dinamiche.

7.2 Definizioni e inizializzazione di variabili puntatore

I puntatori sono variabili i cui valori sono indirizzi di memoria. Normalmente, una variabile contiene *direttamente* un valore specifico. Un pun-

tatore, tuttavia, contiene un indirizzo di un'altra variabile che contiene un valore specifico. Il puntatore *punta a* quella variabile. In questo senso, il nome di una variabile fa riferimento direttamente a un valore, mentre un puntatore fa indirettamente riferimento a un valore, come nel diagramma seguente:



Far riferimento a un valore per mezzo di un puntatore si dice **indirezione**.

Dichiarare i puntatori

I puntatori, come tutte le variabili, devono essere definiti prima di essere utilizzati. L'istruzione seguente definisce la variabile `countPtr` come un `int *` (un puntatore a un intero):

```
int *countPtr;
```

Questa definizione si legge da destra a sinistra, “`countPtr` è un puntatore a un `int`” oppure “`countPtr` punta a un oggetto¹ di tipo `int`”. Il simbolo `*` indica che la variabile è un puntatore.

Denominare le variabili puntatore

La convenzione da noi scelta stabilisce di usare `Ptr` alla fine del nome di ogni variabile puntatore per indicare che la variabile è un puntatore e deve essere trattata di conseguenza. Un'altra comune convenzione di denominazione prevede di iniziare il nome della variabile con `p` (es. `pCount`) oppure `p_` (es. `p_count`).

Definire le variabili in istruzioni separate

Il simbolo `*` nella definizione seguente non viene distribuito a tutte le variabili:

```
int *countPtr, count;
```

quindi, `countPtr` è un puntatore a un intero, ma `count` è soltanto un intero. Per questo motivo, dovete sempre scrivere la dichiarazione precedente nella forma di due istruzioni, per evitare ambiguità:

```
int *countPtr;
int count;
```

Inizializzare e assegnare valori ai puntatori

I puntatori devono essere inizializzati quando sono definiti, oppure assegnando loro un valore. Un puntatore può essere inizializzato a `NULL`, `0` o a un indirizzo.



- Un puntatore con il valore `NULL` non punta a *niente*. `NULL` è una *costante simbolica* con il valore `0` ed è definita nell'intestazione `<stddef.h>` (e in diverse altre intestazioni, come `<stdio.h>`).
- Inizializzare un puntatore a `0` equivale a inizializzare un puntatore a `NULL`. La costante `NULL` è preferibile poiché sottolinea che state inizializzando un puntatore piuttosto che una variabile che memorizza un numero. Quando si assegna `0`, questo viene prima convertito a un puntatore del tipo appropriato. Il valore `0` è l'unico valore intero che si può assegnare direttamente a una variabile puntatore.
- L'assegnazione di un indirizzo di variabile a un puntatore sarà esaminata nel Paragrafo 7.3. Iniziate a inizializzare i puntatori per prevenire risultati inaspettati.

1. In C, un “oggetto” è un’area della memoria che può contenere un valore. Quindi gli oggetti in C includono tipi primitivi come `int`, `float`, `char` e `double`, così come tipi aggregati quali array e strutture (`struct`), che tratteremo nel Capitolo 10.

✓ Autovalutazione

1. (Vero/Falso) La definizione:

```
int *countPtr, count;
```

specifica che countPtr e count sono di tipo int *, ciascuno è un puntatore a un intero.

Risposta: Falso. In realtà, count è un intero, non un puntatore a un intero. Il simbolo * si applica solo a countPtr e non viene distribuito alle altre variabili presenti nella definizione.

2. (Scelta multipla) Quale delle seguenti affermazioni è falsa?

- a) Un puntatore può essere inizializzato a NULL, 0 o a un indirizzo.
- b) Inizializzare un puntatore a 0 equivale a inizializzare un puntatore a NULL, ma 0 è preferibile.
- c) L'unico valore intero che si può assegnare direttamente a una variabile puntatore è 0.
- d) Inizializzate i puntatori per prevenire risultati inaspettati.

Risposta: b) è falsa. In realtà, NULL è preferibile, poiché evidenzia il fatto che la variabile è di un tipo puntatore.

7.3 Operatori per i puntatori

Discussiamo ora gli operatori di indirizzo (&) e di indirezione (*) e la loro relazione.

Operatore di indirizzo (&)

L'operatore di indirizzo (&) unario restituisce l'*indirizzo* del suo operando. Per esempio, data la seguente definizione di y:

```
int y = 5;
```

l'istruzione

```
int *yPtr = &y;
```

inizializza la variabile puntatore yPtr con l'*indirizzo* della variabile y: si dice pertanto che yPtr “punta a” y. Il diagramma seguente mostra le variabili yPtr e y in memoria:



Rappresentazione di un puntatore in memoria

Il diagramma seguente mostra la rappresentazione del puntatore precedente in memoria, supponendo che la variabile intera y sia memorizzata alla locazione 600000 e che la variabile puntatore yPtr sia memorizzata alla locazione 500000:



L'operando di & deve essere una variabile; l'operatore di indirizzo *non può* essere applicato a valori letterali (come 27 o 41.5) o espressioni.

Operatore di indirezione (*)

L'operatore di indirezione (*) unario, anche chiamato **operatore di dereferenziazione**, viene applicato a un operando (un puntatore) per ottenere il *valore* dell'oggetto puntato. Per esempio, la seguente istruzione stampa 5, che è il valore della variabile y:

```
printf("%d", *yPtr);
```

Usare * in questo modo equivale a **dereferenziare un puntatore**.

- ⊗ Dereferenziare un puntatore che non è stato correttamente inizializzato o a cui non è stato assegnato l'indirizzo di un'altra variabile in memoria è un errore. Ciò potrebbe

- determinare un errore irreversibile in fase di esecuzione;
- causare la modifica accidentale di dati importanti e permettere al programma di completare l'esecuzione con risultati scorretti;
- portare a violazioni di sicurezza.²

Come funzionano gli operatori & e *

Il programma della Figura 7.1 illustra l'uso degli operatori & e *. La specifica di conversione %p di printf stampa la locazione di memoria come un intero esadecimale sulla maggior parte delle piattaforme.³ L'output mostra che l'*indirizzo* di a e il *valore* di aPtr sono identici, confermando così che l'indirizzo di a è stato davvero assegnato alla variabile puntatore aPtr (riga 7). Gli operatori & e * sono l'uno il complemento dell'altro: quando entrambi sono applicati consecutivamente ad aPtr in un ordine o nell'altro (riga 12) viene stampato lo stesso risultato. Gli indirizzi mostrati nell'output cambieranno nei vari sistemi che utilizzano architetture del processore, compilatori e persino impostazioni del compilatore differenti.

```
1 // fig07_01.c
2 // Uso degli operatori & e *.
3 #include <stdio.h>
4
5 int main(void) {
6     int a = 7;
7     int *aPtr = &a; // imposta aPtr all'indirizzo di a
8
9     printf("Address of a is %p\nValue of aPtr is %p\n\n", &a, aPtr);
10    printf("Value of a is %d\nValue of *aPtr is %d\n\n", a, *aPtr);
11    printf("Showing that * and & are complements of each other\n");
12    printf("&*aPtr = %p\n*&aPtr = %p\n", &*aPtr, *&aPtr);
13 }
```

```
Address of a is 0x7ffe69386cc
Value of aPtr is 0x7ffe69386cc
Value of a is 7
Value of *aPtr is 7
Showing that * and & are complements of each other
&*aPtr = 0x7ffe69386cc
*&aPtr = 0x7ffe69386cc
```

Figura 7.1 Uso degli operatori & e *.

2. <https://cwe.mitre.org/data/definitions/824.html>.

3. Consultate l'Appendice E (disponibile sulla piattaforma MyLab) per maggiori informazioni sugli interi esadecimali.

La tabella seguente elenca la precedenza e l'associatività degli operatori introdotti fino a questo punto.

Operatori	Associatività	Tipo
) [] ++ (postfisso) -- (postfisso)	da sinistra a destra	postfisso
+ - ++ -- ! * & (tipo)	da destra a sinistra	unario
* / %	da sinistra a destra	moltiplicativo
+ -	da sinistra a destra	additivo
< <= > >=	da sinistra a destra	relazionale
== !=	da sinistra a destra	di uguaglianza
&&	da sinistra a destra	AND logico
	da sinistra a destra	OR logico
? :	da destra a sinistra	condizionale
= += -= *= /= %=	da destra a sinistra	di assegnazione
,	da sinistra a destra	virgola

✓ Autovalutazione

1. (*Vero/Falso*) Presupponendo le definizioni

```
double d = 98.6;
double *dPtr;
```

l'istruzione seguente assegna l'indirizzo della variabile d alla variabile puntatore dPtr:

```
dPtr = &d;
```

Si dice pertanto che la variabile dPtr "punta a" d.

Risposta: *Vero*.

2. (*Completare*) L'operatore unario di indirezione (*) restituisce il valore dell'oggetto al quale punta il suo operando (puntatore). Usare l'operatore * in questo modo viene detto _____.

Risposta: dereferenziare un puntatore.

7.4 Passare argomenti a funzioni per riferimento

Vi sono due modi di passare argomenti a una funzione: **passaggio per valore** e **passaggio per riferimento**. Per default, tutti gli argomenti (che non siano array) sono passati per valore. Come abbiamo visto, gli array sono passati per riferimento. Le funzioni spesso richiedono di poter modificare le variabili nella funzione chiamante o che venga passato un puntatore a un oggetto contenente grandi quantità di dati per evitare il sovraccarico causato dal dover fare una copia dell'oggetto (come avviene nel passaggio per valore). Come abbiamo visto nel Capitolo 5, un'istruzione `return` può restituire al massimo un valore da una funzione chiamata alla sua chiamante. Il passaggio per riferimento si può usare anche per consentire a una funzione di "restituire" più valori modificando le variabili della funzione chiamante.

Usare & e * per realizzare il passaggio per riferimento

I puntatori e l'operatore di indirezione consentono di realizzare il passaggio per riferimento. Quando si chiama una funzione con argomenti che devono essere modificati nella funzione chiamante, si usa l'operatore & per passare l'indirizzo di ogni variabile. Come abbiamo visto nel Capitolo 6, gli array *non* vengono passati usando l'operatore & perché il nome di un array è equivalente a `&arrayNome[0]`, ovvero la locazione iniziale dell'array in memoria. Una funzione che riceve l'indirizzo di una variabile nella funzione chiamante può usare l'operatore di indirezione (*) per modificare il valore in quella locazione nella memoria della funzione chiamante, effettuando così un passaggio per riferimento.

Passaggio per valore

I programmi delle Figure 7.2 e 7.3 presentano due versioni di una funzione che eleva al cubo un intero: `cubeByValue` e `cubeByReference`. Nella riga 11 della Figura 7.2 viene passata per valore la variabile `number` alla funzione `cubeByValue` (righe 16-18), che eleva al cubo il suo argomento e restituisce il nuovo valore. La riga 11 assegna il nuovo valore alla variabile `number` in `main`, sostituendo il valore corrente di `number`.

```

1 // fig07_02.c
2 // Calcolo del cubo di una variabile usando il passaggio per valore,
3 #include <stdio.h>
4
5 int cubeByValue(int n); // prototipo
6
7 int main(void) {
8     int number = 5; // inizializza number
9
10    printf("The original value of number is %d", number);
11    number = cubeByValue(number); // passa number per valore a cubeByValue
12    printf("\nThe new value of number is %d\n", number);
13 }
14
15 // calcola e restituisci il cubo di un argomento intero
16 int cubeByValue(int n) {
17     return n * n * n; // restituisci il cubo di n
18 }
```

```
The original value of number is 5
The new value of number is 125
```

Figura 7.2 Calcolo del cubo di una variabile usando il passaggio per valore.

Passaggio per riferimento

La riga 12 della Figura 7.3 passa l'indirizzo della variabile `number` alla funzione `cubeByReference` (righe 17-19): il passaggio dell'indirizzo permette il passaggio per riferimento. Il parametro della funzione è un puntatore a un `int` chiamato `nPtr` (riga 17). La funzione usa l'espressione `*nPtr` per dereferenziare il puntatore ed elevare al cubo il valore al quale esso punta (riga 18). Assegna il risultato a `*nPtr` (che è in effetti la variabile `number` in `main`), cambiando così il valore di `number` in `main`. Usate il passaggio per valore, a meno che la funzione chiamante non necessiti esplicitamente che la funzione chiamata modifichi il valore della variabile argomento nella stessa funzione chiamante. Ciò previene la modifica accidentale degli argomenti passati dalla funzione chiamante e costituisce inoltre un ulteriore esempio del principio del privilegio minimo.



```

1 // fig07_03.c
2 // Calcolo del cubo di una variabile usando il passaggio per riferimento.
3
4 #include <stdio.h>
5
6 void cubeByReference(int *nPtr); // prototipo di funzione
7
8 int main(void) {
9     int number = 5; // inizializza number
10
11    printf("The original value of number is %d", number);
12    cubeByReference(&number); // passa l'indirizzo di number a cubeByReference
13    printf("\nThe new value of number is %d\n", number);
14 }
15
16 // eleva al cubo *nPtr; di fatto modifica number in main
17 void cubeByReference(int *nPtr) {
18     *nPtr = *nPtr * *nPtr * *nPtr; // calcola il cubo di *nPtr
19 }
```

The original value of number is 5
 The new value of number is 125

Figura 7.3 Calcolo del cubo di una variabile usando il passaggio per riferimento con un argomento puntatore.

Usare un parametro puntatore per ricevere un indirizzo

Una funzione che riceve un indirizzo come argomento deve riceverlo con un parametro puntatore. Per esempio, nel programma della Figura 7.3, l'intestazione per la funzione cubeByReference (riga 17) è

```
void cubeByReference(int *nPtr) {
```

e specifica che cubeByReference riceve l'indirizzo di una variabile intera come argomento, memorizza l'indirizzo localmente nel parametro nPtr e non restituisce alcun valore.

Parametri puntatore nei prototipi di funzioni

Il prototipo di funzione per cubeByReference (Figura 7.3, riga 6) specifica un parametro `int *`. Come con altri parametri, non è necessario includere i nomi dei puntatori nei prototipi di funzioni (essi vengono ignorati dal compilatore) ma è buona pratica includerli per fini di documentazione.

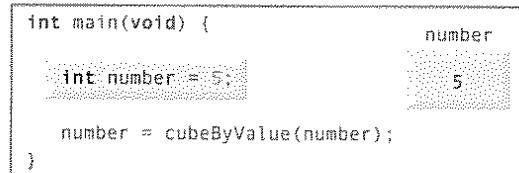
Funzioni che ricevono array unidimensionali

Per una funzione che si aspetta come argomento un array unidimensionale, il prototipo e l'intestazione della funzione possono usare la notazione usata per i puntatori mostrata nella lista dei parametri della funzione cubeByReference (riga 17). Il compilatore non distingue tra una funzione che riceve un puntatore e una che riceve un array unidimensionale. Ciò, naturalmente, significa che la funzione deve “sapere” quando riceve un array o semplicemente una variabile singola per la quale si sta eseguendo il passaggio per riferimento. Quando il compilatore incontra un parametro di funzione per un array unidimensionale della forma `int b[]`, il compilatore converte il parametro nella notazione per i puntatori `int *b`. Le due forme sono intercambiabili. Analogamente, per un parametro della forma `const int b[]` il compilatore converte il parametro in `const int *b`.

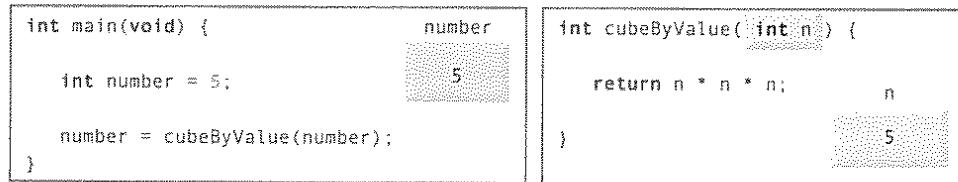
Analisi grafica del passaggio per valore e del passaggio per riferimento

Le Figure 7.4 e 7.5 analizzano graficamente passo per passo i programmi, rispettivamente, delle Figure 7.2 e 7.3.

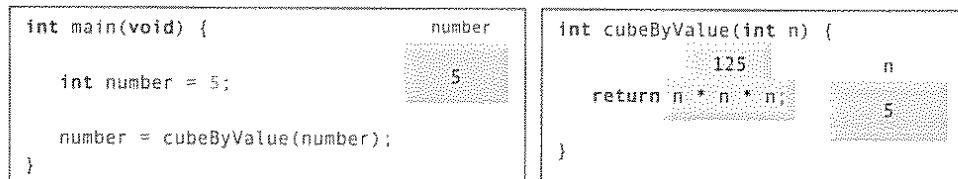
Passo 1: Prima che main chiami cubeByValue:



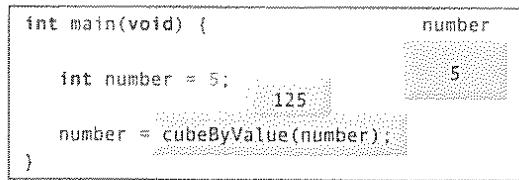
Passo 2: Dopo che cubeByValue ha ricevuto la chiamata:



Passo 3: Dopo che cubeByValue ha elevato al cubo il parametro n e prima che cubeByValue torni alla funzione main:



Passo 4: Dopo che cubeByValue è tornata alla funzione main e prima che si assegni il risultato a number:



Passo 5: Dopo che main ha completato l'assegnazione a number:

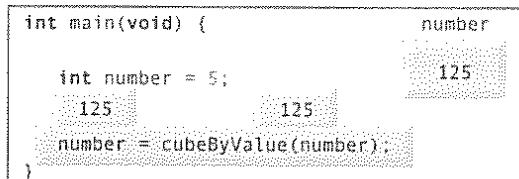
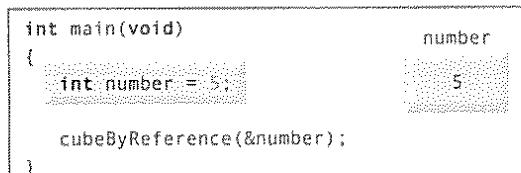
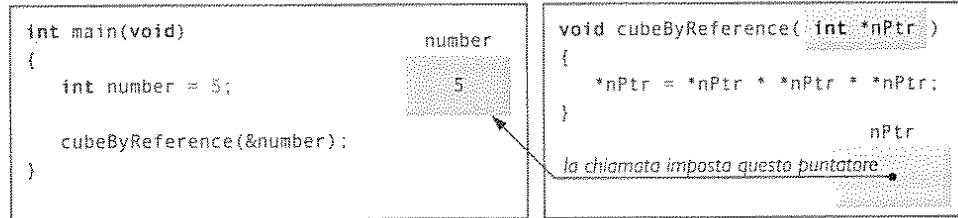


Figura 7.4 Analisi di un tipico passaggio per valore.

Passo 1: Prima che main chiami cubeByReference:



Passo 2: Dopo che cubeByReference ha ricevuto la chiamata e prima che *nPtr sia elevato al cubo:



Passo 3: Dopo che *nPtr è stato elevato al cubo e prima che il controllo del programma torni alla funzione main:

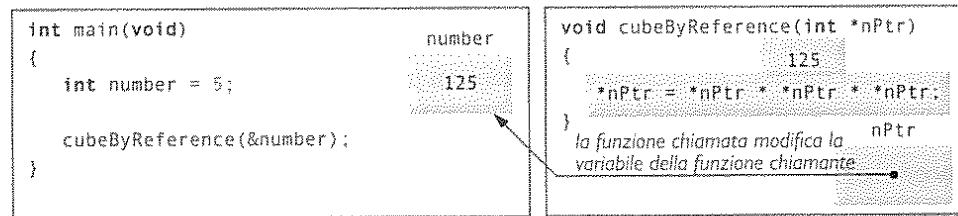


Figura 7.5 Analisi di un tipico passaggio per riferimento con un argomento puntatore.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- Per default, tutti gli argomenti (che non siano array) sono passati per valore. Gli array sono passati per riferimento.
- Le funzioni spesso richiedono di poter modificare le variabili nella funzione chiamante o che venga passato un puntatore a un oggetto contenente grandi quantità di dati per evitare di fare una copia dell'oggetto.
- L'istruzione `return` si può usare per restituire uno o più valori da una funzione chiamata a una chiamante.
- Il passaggio per riferimento si può usare anche per consentire a una funzione di “restituire” più valori modificando variabili nella funzione chiamante.

Risposta: c) è *falsa*. L'istruzione `return` si può usare per restituire al massimo un valore da una funzione chiamata a una chiamante.

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- Si usano i puntatori e l'operatore di indirezione per realizzare il passaggio per riferimento.
- Quando si chiama una funzione con argomenti che devono essere modificati, si usa l'operatore di indirizzo (&) per passare gli indirizzi degli argomenti.
- Gli array vengono passati per riferimento usando l'operatore &.
- Tutte le affermazioni precedenti sono *vere*.

Risposta: c) è *falsa*. Gli array non vengono passati per riferimento usando l'operatore & perché il nome di un array è equivalente all'indirizzo del suo primo elemento: `&arrayNome[0]`.

7.5 Uso del qualificatore `const` con i puntatori

 Il qualificatore `const` permette di informare il compilatore che il valore di una particolare variabile non deve essere modificato, applicando così il principio del privilegio minimo. Ciò può ridurre il tempo di debugging ed evitare effetti secondari indesiderati, rendendo un programma più robusto e facile da modificare e mantenere. Se si cerca di modificare un valore dichiarato `const`, il compilatore lo rileva ed emette un messaggio di errore.

 Nel corso degli anni è stata scritta una grande quantità di codice vecchio stile o cosiddetto legacy (letteralmente “ereditato”) nelle prime versioni del C che non usavano `const` perché non era disponibile. Anche in codice più recente non viene usato `const` tanto quanto si dovrebbe. Per questo motivo esistono importanti opportunità di miglioramento tramite la reingegnerizzazione del codice in C esistente.

Vi sono quattro modi per passare a una funzione un puntatore a dati:

- un puntatore non costante a dati non costanti;
- un puntatore costante a dati non costanti;
- un puntatore non costante a dati costanti;
- un puntatore costante a dati costanti.

Ciascuna delle quattro combinazioni concede differenti privilegi d’accesso che saranno esaminati nei prossimi vari esempi. Come scegliere una delle possibili opzioni? Lasciatevi guidare dal principio del privilegio minimo. Date sempre a una funzione un accesso ai dati nei suoi parametri sufficiente per portare a termine il suo compito specifico, ma assolutamente null’altro.

7.5.1 Conversione di una stringa in maiuscolo usando un puntatore non costante a dati non costanti

Il livello più alto di accesso ai dati è concesso da un puntatore non costante a dati non costanti. I dati possono essere modificati per mezzo del puntatore dereferenziato e il puntatore può essere modificato per puntare ad altri dati. Una funzione può usare un puntatore di questo genere per ricevere una stringa come argomento, quindi elaborare (ed eventualmente modificare) ogni carattere nella stringa. La funzione `convertToUppercase` nella Figura 7.6 dichiara il suo parametro, un puntatore non costante a dati non costanti chiamato `sPtr` (riga 18). La funzione elabora l’array `string` (puntato da `sPtr`) un carattere alla volta. La funzione della Libreria Standard del C `toupper` (riga 20) dichiarata nel file di intestazione `<ctype.h>` converte ogni carattere nel formato maiuscolo corrispondente. Se il carattere originario non è una lettera o è già in maiuscolo, `toupper` restituisce il carattere originario. La riga 21 incrementa il puntatore per puntare al carattere successivo nella stringa. Il Capitolo 8 presenta molte funzioni per l’elaborazione di caratteri e stringhe della Libreria Standard del C.

```

1 // fig07_06.c
2 // Conversione di una stringa in maiuscolo usando
3 // un puntatore non costante a dati non costanti.
4 #include <ctype.h>
5 #include <stdio.h>
6
7 void convertToUppercase(char *sPtr); // prototipo
8
9 int main(void) {
10     char string[] = "cHaRaCters and $32.98"; // inizializza l'array di caratteri
11
12     printf("The string before conversion is: %s\n", string);
13     convertToUppercase(string);
14     printf("The string after conversion is: %s\n", string);
15 }
16
17 // converti la stringa in lettere maiuscole

```

```

18 void convertToUppercase(char *sPtr) {
19     while (*sPtr != '\0') { // il carattere corrente non e' '\0'
20         *sPtr = toupper(*sPtr); // converti in maiuscolo
21         ++sPtr; // fai puntare sPtr al carattere successivo
22     }
23 }
```

The string before conversion is: cHaRaCters and \$32.98
The string after conversion is: CHARTERS AND \$32.98

Figura 7.6 Conversione di una stringa in maiuscolo usando un puntatore non costante a dati non costanti.

7.5.2 Stampa di una stringa un carattere alla volta usando un puntatore non costante a dati costanti

Un **puntatore non costante a dati costanti** può essere modificato per puntare a un elemento qualunque di dati del tipo appropriato, ma i dati a cui punta *non possono essere modificati*. Una funzione può ricevere un puntatore di questo tipo per elaborare gli elementi di un argomento array senza modificarli. Per esempio, la funzione printCharacters (Figura 7.7) dichiara il parametro sPtr di tipo `const char *` (riga 20). La dichiarazione va letta da *destra a sinistra* come “sPtr è un puntatore a una costante carattere”. L’istruzione `for` della funzione stampa ogni carattere nella stringa finché non incontra il carattere nullo. Dopo la stampa di ogni carattere, il ciclo incrementa il puntatore sPtr in modo che punti al carattere successivo nella stringa.

```

1 // fig07_07.c
2 // Stampa di una stringa un carattere alla volta usando
3 // un puntatore non costante a dati costanti.
4
5 #include <stdio.h>
6
7 void printCharacters(const char *sPtr);
8
9 int main(void) {
10     // inizializza l'array di caratteri
11     char string[] = "print characters of a string";
12
13     puts("The string is:");
14     printCharacters(string);
15     puts("");
16 }
17
18 // sPtr non puo' essere usato per modificare il carattere al quale punta,
19 // cioe', sPtr e' un puntatore di "sola lettura"
20 void printCharacters(const char *sPtr) {
21     // effettua un ciclo lungo l'intera stringa
22     for (; *sPtr != '\0'; ++sPtr) { // nessuna inizializzazione
23         printf("%c", *sPtr);
24     }
25 }
```

The string is:
print characters of a string

Figura 7.7 Stampa di una stringa un carattere alla volta usando un puntatore non costante a dati costanti.

Tentativo di modifica di dati costanti

Il programma della Figura 7.8 illustra gli errori derivanti dal compilare una funzione che riceve un puntatore non costante (`xPtr`) a dati costanti e tenta di usarlo per modificare i dati. L'errore mostrato di seguito proviene dal compilatore Visual C++. Il C standard non specifica il tipo di messaggi di avvertimento o di errore del compilatore, e i fornitori di compilatori non usano messaggi uniformi per i diversi compilatori. Quindi il particolare messaggio di errore che si riceve dipende dal compilatore. Per esempio, il compilatore LLVM di Xcode genera il seguente messaggio:

```
error: read-only variable is not assignable
```

e il compilatore GNU gcc genera il messaggio:

```
error: assignment of read-only location '*xPtr'  
1 // fig07_08.c  
2 // Tentativo di modifica di dati con un  
3 // puntatore non costante a dati costanti.  
4 #include <stdio.h>  
5 void f(const int *xPtr); // prototipo  
6  
7 int main(void) {  
8     int y = 7; // definisci y  
9  
10    f(&y); // f tenta una modifica non permessa  
11 }  
12  
13 // xPtr non puo' essere usato per modificare  
14 // il valore della variabile alla quale punta  
15 void f(const int *xPtr) {  
16     *xPtr = 100; // errore: non si puo' modificare un oggetto const  
17 }
```

Messaggio di errore di Visual C++ di Microsoft

```
fig07_08.c(16,5): error C2166: l-value specifies const object
```

Figura 7.8 Tentativo di modifica di dati con un puntatore non costante a dati costanti.

Passare strutture vs array

Com'è noto, gli array sono tipi di dati aggregati che memorizzano elementi di dati correlati dello stesso tipo sotto un solo nome. Nel Capitolo 10 esamineremo un'altra forma di tipo di dati aggregati chiamata **struttura** (talvolta chiamata **record** o **tupla** in altri linguaggi), che è in grado di memorizzare elementi di dati correlati dello stesso tipo o di tipi *differenti* sotto un unico nome (es. informazioni relative a un impiegato, come numero di matricola, nome, indirizzo e stipendio).

A differenza degli array, le strutture vengono passate per valore (viene passata una copia dell'intera struttura). Ciò richiede ulteriore tempo di esecuzione (overhead) per fare una copia di ogni elemento dei dati nella struttura e per memorizzarla nella pila delle chiamate di funzione del computer. Passare oggetti di grandi dimensioni come le strutture utilizzando puntatori a dati costanti consente di ottenere le prestazioni del passaggio per riferimento e la sicurezza del passaggio per valore. In questo caso, il programma fa una copia solo dell'*indirizzo* nel quale la struttura è memorizzata (la copia occupa in genere quattro o otto byte).

Se la memoria è poca e l'efficienza dell'esecuzione è un problema, usate i puntatori. Se la memoria è molta e l'efficienza non è un grosso problema, passate i dati per valore per applicare il principio del privilegio minimo. Alcuni sistemi non applicano bene `const`, per cui il passaggio per valore è ancora il modo migliore per evitare che i dati vengano modificati.

7.5.3 Tentativo di modificare un puntatore costante a dati non costanti

Un puntatore costante a dati non costanti punta sempre alla stessa locazione di memoria, ma i dati in quella locazione *possono essere modificati* per mezzo del puntatore. Questa è l'impostazione predefinita per il nome di un array, che è un puntatore costante al primo elemento dell'array. È possibile accedere a tutti i dati nell'array e modificarli usando il nome e l'indice dell'array, ed è possibile usare un puntatore costante a dati non costanti per passare un array come argomento a una funzione che accede agli elementi dell'array usando solo la notazione con indice. I puntatori che sono dichiarati `const` devono essere inizializzati quando sono definiti. Se il puntatore è il parametro di una funzione, è inizializzato con un argomento puntatore quando la funzione viene chiamata.

Il programma della Figura 7.9 tenta di modificare un puntatore costante. Il puntatore `ptr` è definito nella riga 11 di tipo `int * const`, che si legge *da destra a sinistra* come “`ptr` è una costante che punta a un intero”. Il puntatore è inizializzato (riga 11) con l'indirizzo della variabile intera `x`. Il programma tenta di assegnare l'indirizzo di `y` a `ptr` (riga 14), ma il compilatore genera un errore.

```

1 // fig07_09.c
2 // Tentativo di modificare un puntatore costante a dati non costanti.
3 #include <stdio.h>
4
5 int main(void) {
6     int x = 0; // definisci x
7     int y = 0; // definisci y
8
9     // ptr e' un puntatore costante a un intero che puo' essere modificato
10    // tramite ptr, ma ptr punta sempre alla stessa locazione di memoria
11    int * const ptr = &x;
12
13    *ptr = 7; // permesso: *ptr non e' const
14    ptr = &y; // errore: ptr e' const; non puo' assegnare un nuovo indirizzo
15 }
```

Messaggio di errore di Visual C++ di Microsoft

fig07_09.c(14,4): error C2166: l-value specifies const object

Figura 7.9 Tentativo di modificare un puntatore costante a dati non costanti.

7.5.4 Tentativo di modificare un puntatore costante a dati costanti

Il privilegio di accesso *più basso* è quello concesso da un **puntatore costante a dati costanti**. Un tale puntatore punta sempre alla stessa locazione di memoria e i dati in quella locazione di memoria *non possono essere modificati*. Questo è il modo in cui un array deve essere passato a una funzione che legge soltanto l'array usando la notazione con indice e *non* modifica gli elementi. Il programma della Figura 7.10 definisce la variabile puntatore `ptr` (riga 12) di tipo `const int *const`, che si legge *da destra a sinistra* come “`ptr` è un puntatore costante a un intero costante”. L'output mostra il messaggio di errore generato quando si tenta di *modificare i dati* ai quali punta `ptr` (riga 15) e quando si tenta di *modificare l'indirizzo* memorizzato nella variabile puntatore (riga 16).

```

1 // fig07_10.c
2 // Tentativo di modificare un puntatore costante a dati costanti.
3 #include <stdio.h>
4
5 int main(void) {
6     int x = 5;
```

```

7     int y = 0;
8
9     // ptr e' un puntatore costante a un intero costante. ptr
10    // punta sempre alla stessa locazione; l'intero in quella locazione
11    // non puo' essere modificato
12    const int *const ptr = &x; // l'inizializzazione e' OK
13
14    printf("%d\n", *ptr);
15    *ptr = 7; // errore: *ptr e' const; non puo' assegnare un nuovo valore
16    ptr = &y; // errore: ptr e' const; non puo' assegnare un nuovo indirizzo
17 }

```

Messaggio di errore di Visual C++ di Microsoft

```

fig07_10.c(15,5): error C2166: l-value specifies const object
fig07_10.c(16,4): error C2166: l-value specifies const object

```

Figura 7.10 Tentativo di modificare un puntatore costante a dati costanti.

✓ Autovalutazione

1. (*Scelta multipla*) Cosa rappresenta sPtr nel seguente prototipo?

```
void convertToUppercase(char *sPtr);
```

- a) Un puntatore non costante a dati costanti.
- b) Un puntatore costante a dati non costanti.
- c) Un puntatore non costante a dati non costanti.
- d) Un puntatore costante a dati costanti.

Risposta: c.

2. (*Completare*) Il privilegio di accesso più basso è quello concesso da un puntatore _____ a dati _____. Un puntatore di questo tipo punta sempre alla stessa locazione di memoria e i dati in quella locazione di memoria non possono essere modificati.

Risposta: costante, costanti.

7.6 Bubble sort che utilizza il passaggio per riferimento

Miglioriamo il programma per il bubble sort⁴ della Figura 6.12 per usare due funzioni: bubbleSort e swap (Figura 7.11). La funzione bubbleSort ordina l'array e chiama la funzione swap (riga 42) per scambiare tra loro gli elementi array[j] e array[j + 1].

```

1 // fig07_11.c
2 // Inserimento di valori in un array, ordinamento dei valori
3 // in ordine crescente e stampa dell'array risultante.
4 #include <stdio.h>
5 #define SIZE 10
6
7 void bubbleSort(int * const array, size_t size); // prototipo
8
9 int main(void) {

```

4. Nei Capitoli 12 e 13 esaminiamo gli schemi di ordinamento che producono le prestazioni migliori.

```

10 // inizializza l'array a
11 int a[SIZE] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
12
13 puts("Data items in original order");
14
15 // effettua un ciclo lungo l'array a
16 for (size_t i = 0; i < SIZE; ++i) {
17     printf("%4d", a[i]);
18 }
19
20 bubbleSort(a, SIZE); // ordina l'array
21
22 puts("\nData items in ascending order");
23
24 // effettua un ciclo lungo l'array a
25 for (size_t i = 0; i < SIZE; ++i) {
26     printf("%4d", a[i]);
27 }
28
29 puts("");
30 }
31
32 // ordina un array di interi usando l'algoritmo bubble sort
33 void bubbleSort(int * const array, size_t size) {
34     void swap(int *element1Ptr, int *element2Ptr); // prototipo
35
36     // ciclo di controllo delle passate
37     for (int pass = 0; pass < size - 1; ++pass) {
38         // ciclo di controllo dei confronti durante ogni passata
39         for (size_t j = 0; j < size - 1; ++j) {
40             // scambia gli elementi adiacenti se non sono in ordine
41             if (array[j] > array[j + 1]) {
42                 swap(&array[j], &array[j + 1]);
43             }
44         }
45     }
46 }
47
48 // scambia i valori delle locazioni di memoria alle quali element1Ptr
49 // ed element2Ptr rispettivamente puntano
50 void swap(int *element1Ptr, int *element2Ptr) {
51     int hold = *element1Ptr;
52     *element1Ptr = *element2Ptr;
53     *element2Ptr = hold;
54 }
```

```

Data items in original order
 2  6  4  8 10 12 89 68 45 37
Data items in ascending order
 2  4  6  8 10 12 37 45 68 89
```

Figura 7.11 Inserimento di valori in un array, ordinamento dei valori in ordine crescente e stampa dell'array risultante.

Funzione swap

Ricordate che il C applica il principio di *occultamento delle informazioni* alle interazioni tra funzioni, per cui swap non ha accesso ai singoli elementi dell'array in bubbleSort per default. Poiché bubbleSort vuole che swap abbia accesso agli elementi dell'array da scambiare, bubbleSort passa l'indirizzo di ogni elemento a swap, quindi gli elementi sono passati per riferimento. Sebbene gli array nella loro interezza siano passati automaticamente per riferimento, i loro singoli elementi sono *scalari* e sono ordinariamente passati per valore. Pertanto, bubbleSort usa l'operatore di indirizzo (&) su ogni elemento dell'array:

```
swap(&array[j], &array[j + 1]);
```

La funzione swap riceve &array[j] in element1Ptr (riga 50). La funzione swap può usare *element1Ptr come sinonimo di array[j]. Allo stesso modo, *element2Ptr è un sinonimo di array[j + 1]. Anche se nel corpo della funzione swap non è permesso scrivere

```
int hold = array[j];
array[j] = array[j + 1];
array[j + 1] = hold;
```

si ottiene esattamente lo stesso effetto con le righe dalla 51 alla 53:

```
int hold = *element1Ptr;
*element1Ptr = *element2Ptr;
*element2Ptr = hold;
```

Parametro array della funzione bubbleSort

Notate che l'intestazione della funzione bubbleSort (riga 33) dichiara array come `int * const array` invece che come `int array[]` per indicare che bubbleSort riceve un array unidimensionale come argomento. Ancora, queste notazioni sono intercambiabili; tuttavia, le notazioni con array sono di norma preferibili per una migliore leggibilità.

Prototipo della funzione swap nel corpo della funzione bubbleSort

Il prototipo per la funzione swap (riga 34) è incluso nel corpo della funzione bubbleSort perché bubbleSort è l'unica funzione che chiama swap. Mettere il prototipo in bubbleSort riduce le chiamate appropriate di swap solo a quelle fatte da bubbleSort (o qualsiasi funzione che appaia dopo swap nel codice sorgente). Altre funzioni definite prima di swap che tentano di chiamare swap non hanno accesso al prototipo proprio della funzione, così il compilatore ne genera automaticamente uno. Ciò normalmente produce un prototipo che non corrisponde all'intestazione della funzione (il che genera un avvertimento o un errore di compilazione), perché il compilatore presume `int` come tipo del valore di ritorno e dei parametri. Mettere i prototipi di funzioni nelle definizioni di altre funzioni è un'applicazione del principio del privilegio minimo, perché ciò limita le chiamate appropriate di una funzione alle funzioni in cui compaiono i prototipi.

Parametro size della funzione bubbleSort

La funzione bubbleSort riceve la dimensione dell'array come parametro (riga 33). Quando un array viene passato a una funzione, l'indirizzo di memoria del primo elemento dell'array, ovviamente, non riporta il numero di elementi dell'array. Pertanto, si deve passare alla funzione la dimensione dell'array per comunicare il numero di elementi da ordinare. Un'altra pratica comune è quella di passare un puntatore al primo elemento dell'array e un puntatore alla locazione appena oltre la fine dell'array. Come imparerete nel Paragrafo 7.8, la differenza tra questi due puntatori è la lunghezza dell'array, e il codice che ne risulta è più semplice.

 Vi sono due vantaggi principali nel passare la dimensione dell'array a bubbleSort: *la riusabilità del software* e *la corretta applicazione dei principi dell'ingegneria del software*. Definendo la funzione in modo che essa riceva la dimensione dell'array come argomento, facciamo sì che la funzione possa essere usata da un qualunque programma che debba ordinare array interi con singolo indice di qualsiasi dimensione.

 Avremmo potuto memorizzare la dimensione dell'array in una variabile globale accessibile all'intero programma. Tuttavia, altri programmi che richiedono di ordinare array interi potrebbero non avere la stessa variabile globale, per cui la funzione non può essere usata in quei programmi. Le variabili globali solitamente violano il principio del privilegio minimo e possono portare a una ingegneria del software di bassa qualità. Le

variabili globali vanno usate solo per rappresentare risorse veramente condivise, come per esempio il valore dell'ora del giorno.

La dimensione dell'array potrebbe essere stata programmata direttamente nella funzione. Ciò limiterebbe l'uso della funzione all'elaborazione di un array di una dimensione specifica e ne ridurrebbe significativamente la riutilizzabilità. Solo i programmi che elaborano array interi unidimensionali della dimensione specifica codificata nella funzione potrebbero usare quest'ultima.

✓ Autovalutazione

1. (*Codice*) Nel nostro esempio, la funzione bubbleSort usa l'operatore di indirizzo (&) con ognuno degli elementi dell'array nella chiamata di swap per effettuare il passaggio per riferimento come segue:

```
swap(&array[j], &array[j + 1]);
```

Supponete che la funzione swap riceva &array[j] e &array[j + 1] in puntatori int * chiamati rispettivamente firstPtr e secondPtr. Scrivete il codice in base ai puntatori nella funzione swap per scambiare i valori in questi due elementi usando la variabile int temporanea temp.

Risposta:

```
int temp = *firstPtr;
*firstPtr = *secondPtr;
*secondPtr = temp;
```

2. (*Discussione*) Di norma, quando si passa un array a una funzione, si passa anche come argomento la dimensione dell'array. In alternativa, si potrebbe impostare la dimensione dell'array direttamente nella definizione della funzione. Cosa c'è di sbagliato in questo approccio?

Risposta: Limiterebbe la funzione, che potrebbe elaborare soltanto array di una dimensione specifica, riducendone significativamente la riutilizzabilità.

7.7 Operatore sizeof

Il C fornisce l'operatore unario **sizeof** per determinare la dimensione in byte di un oggetto o di un tipo. Questo operatore viene applicato al momento della compilazione, a meno che il suo operando sia un array di lunghezza variabile (VLA; Paragrafo 6.12). Quando è applicato al nome di un array come nella Figura 7.12 (riga 12), sizeof restituisce come valore di tipo **size_t** il numero totale di byte nell'array. Le variabili di tipo float sul nostro computer sono memorizzate in 4 byte di memoria e array è definito di 20 elementi. Vi sono quindi in totale 80 byte in array. sizeof è un operatore della fase di compilazione, quindi non causa alcun sovraccarico riguardo al tempo di esecuzione (tranne che per i VLA).

```
1 // fig07_12.c
2 // L'applicazione di sizeof al nome di un array restituisce
3 // il numero di byte nell'array.
4 #include <stdio.h>
5 #define SIZE 20
6
7 size_t getSize(const float *ptr); // prototipo
8
9 int main(void){
10     float array[SIZE]; // crea l'array
11
12     printf("Number of bytes in the array is %zu\n", sizeof(array));
13     printf("Number of bytes returned by getSize is %zu\n", getSize(array));
14 }
15
16 // restituisce la dimensione di ptr
```

```
17 size_t getSize(const float *ptr) {
18     return sizeof(ptr);
19 }
```

Number of bytes in the array is 80
Number of bytes returned by getSize is 8

Figura 7.12 L'applicazione di `sizeof` al nome di un array restituisce il numero di byte nell'array.

Anche se la funzione `getSize` riceve un array di 20 elementi come argomento, il parametro `ptr` della funzione è semplicemente un puntatore al primo elemento dell'array. Quando usate `sizeof` con un puntatore, esso restituisce la *dimensione del puntatore*, non la dimensione dell'elemento al quale punta. Sui nostri sistemi di test Mac, Linux e Windows a 64 bit, la dimensione di un puntatore è otto byte, per cui `getSize` restituisce 8. Sui sistemi più vecchi a 32 bit, la dimensione di un puntatore è tipicamente quattro byte, per cui `getSize` restituisce 4.

Anche il numero degli elementi in un array si può determinare con `sizeof`. Per esempio, considerate la seguente definizione di array:

```
double real[22];
```

Le variabili di tipo double sono normalmente memorizzate in 8 byte di memoria. Pertanto, l'array real contiene 176 byte. L'espressione seguente determina il numero di elementi dell'array:

`sizeof(real) / sizeof(real[0])`

L'espressione divide il numero di byte dell'array `real` per il numero dei byte usati per memorizzare il primo elemento dell'array (un valore `double`). Questo calcolo funziona *solo* quando si usa il nome dell'array vero e proprio, *non* quando si usa un puntatore all'array.

Determinare le dimensioni dei tipi standard, di un array e di un puntatore

Il programma della Figura 7.13 calcola il numero di byte usati per memorizzare ognuno dei tipi standard. I risultati di questo programma sono dipendenti dall'implementazione e spesso differiscono fra le varie piattaforme e talvolta fra i diversi compilatori sulla stessa piattaforma. L'output mostra i risultati del nostro sistema Mac usando il compilatore C++ di Xcode.

```
1 // fig07_13.c
2 // Uso dell'operatore sizeof per determinare le dimensioni di tipi standard.
3 #include <stdio.h>
4
5 int main(void) {
6     char c = ;
7     short s = 0;
8     int i = 0;
9     long l = 0;
10    long long ll = 0;
11    float f = 0.0F;
12    double d = 0.0;
13    long double ld = 0.0;
14    int array[20] = {0}; // crea un array di 20 elementi int
15    int *ptr = array; // crea un puntatore all'array
16
17    printf("    sizeof c = %zu\t      sizeof(char) = %zu\n",
18          sizeof c, sizeof(char));
19    printf("    sizeof s = %zu\t      sizeof(short) = %zu\n",
20          sizeof s, sizeof(short));
```

```

21   printf("  sizeof i = %zu\n"      sizeof(int) = %zu\n",
22     sizeof i, sizeof(int));
23   printf("  sizeof l = %zu\n"      sizeof(long) = %zu\n",
24     sizeof l, sizeof(long));
25   printf("  sizeof ll = %zu\n"    sizeof(long long) = %zu\n",
26     sizeof ll, sizeof(long long));
27   printf("  sizeof f = %zu\n"     sizeof(float) = %zu\n",
28     sizeof f, sizeof(float));
29   printf("  sizeof d = %zu\n"     sizeof(double) = %zu\n",
30     sizeof d, sizeof(double));
31   printf("  sizeof ld = %zu\n"    sizeof(long double) = %zu\n",
32     sizeof ld, sizeof(long double));
33   printf("sizeof array = %zu\n sizeof ptr = %zu\n",
34     sizeof array, sizeof ptr);
35 }

```

sizeof c = 1	sizeof(char) = 1
sizeof s = 2	sizeof(short) = 2
sizeof i = 4	sizeof(int) = 4
sizeof l = 8	sizeof(long) = 8
sizeof ll = 8	sizeof(long long) = 8
sizeof f = 4	sizeof(float) = 4
sizeof d = 8	sizeof(double) = 8
sizeof ld = 16	sizeof(long double) = 16
sizeof array = 80	
sizeof ptr = 8	

Figura 7.13 Uso dell'operatore sizeof per determinare le dimensioni di tipi standard.

 Il numero di byte usati per memorizzare un particolare tipo può variare tra i sistemi di computer. Quando scrivete programmi che dipendono dalle dimensioni dei tipi e che saranno eseguiti su diversi sistemi, usate sizeof per determinare il numero di byte usati per memorizzare i vari tipi.

L'operatore sizeof può essere applicato a un qualsiasi nome di variabile, tipo o valore (compreso il valore di un'espressione). Quando è applicato al nome di una variabile (che *non* è il nome di un array) o a una costante, viene restituito il numero di byte usati per memorizzare il tipo specifico della variabile o della costante. Le parentesi sono necessarie quando l'operando di sizeof è un tipo di dati.

✓ Autovalutazione

1. (*Completare*) Data la definizione dell'array:

```
double temperatures[31];
```

l'espressione:

```
sizeof(temperatures) / sizeof(temperatures[0])
```

quale attributo di temperatures determina? _____

Risposta: Il numero di elementi nell'array (in questo caso, 31).

2. (*Vero/Falso*) Quando usate sizeof con un puntatore, esso restituisce la dimensione dell'elemento al quale il puntatore punta.

Risposta: *Falso*. In realtà, quando usate sizeof con un puntatore, esso restituisce la dimensione del puntatore, non la dimensione dell'elemento al quale il puntatore punta. Se usate sizeof con il nome di un array, restituisce la dimensione dell'array.

7.8 Espressioni con puntatori e aritmetica dei puntatori

I puntatori sono operandi validi nelle espressioni aritmetiche, nelle espressioni di assegnazione e nelle espressioni di confronto. Tuttavia, non tutti gli operatori normalmente usati in queste espressioni sono validi in combinazione con le variabili puntatore. Questo paragrafo descrive gli operatori che possono avere puntatori come operandi e il modo in cui tali operatori possono essere usati.

7.8.1 Operatori per l'aritmetica dei puntatori

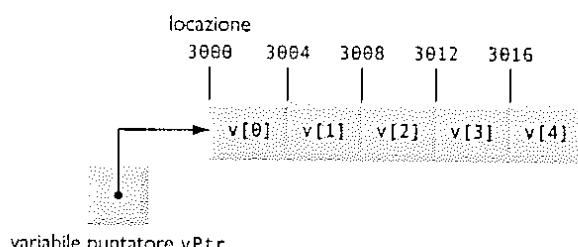
È possibile eseguire le seguenti operazioni aritmetiche con i puntatori:

- incrementare (++) o decrementare (--);
- aggiungere un intero a un puntatore (+ o +=);
- sottrarre un intero da un puntatore (- o -=);
- sottrarre un puntatore da un altro puntatore (operazione che ha senso solo quando *entrambi* i puntatori puntano agli elementi dello stesso array).

 Usare l'aritmetica dei puntatori su puntatori che non fanno riferimento a elementi in un array è un errore logico.

7.8.2 Indirizzare un puntatore a un array

Supponete che sia stato definito l'array `int v[5]` e che il suo primo elemento sia alla locazione 3000 nella memoria. Supponete inoltre che il puntatore `vPtr` punti a `v[0]` (ossia il valore di `vPtr` è 3000). Il diagramma seguente illustra questo scenario per una macchina con interi di 4 byte:



La variabile `vPtr` può essere inizializzata per puntare all'array `v` con l'una o l'altra delle istruzioni

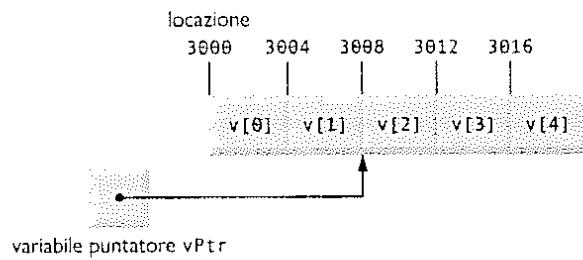
```
vPtr = v;
vPtr = &v[0];
```

7.8.3 Aggiungere un intero a un puntatore

Nell'aritmetica convenzionale, $3000 + 2$ produce il valore 3002. Questo non avviene normalmente con l'aritmetica dei puntatori. Quando un intero è sommato a un puntatore o sottratto da esso, il puntatore è incrementato o decrementato di quell'intero *moltiplicato per la dimensione dell'oggetto a cui il puntatore fa riferimento*. Per esempio, l'istruzione

```
vPtr += 2;
```

darebbe il valore 3008 ($3000 + 2 * 4$), supponendo che un intero sia memorizzato in 4 byte di memoria. Nell'array `v`, `vPtr` punterebbe adesso a `v[2]`, come nel seguente diagramma:



La dimensione di un oggetto dipende dal tipo dell'oggetto stesso. Quando si usa l'aritmetica dei puntatori con un array di caratteri, i risultati saranno coerenti con l'aritmetica regolare perché ogni carattere è lungo 1 byte. Le dimensioni dei tipi possono variare in base alla piattaforma e al compilatore, quindi l'aritmetica dei puntatori dipende dalla piattaforma e dal compilatore.

7.8.4 Sottrarre un intero da un puntatore

Se `vPtr` fosse stato incrementato fino a 3016 (`v[4]`), l'istruzione

```
vPtr -= 4;
```

riporterebbe `vPtr` al valore 3000 (`v[0]`), cioè all'inizio dell'array. Usare l'aritmetica dei puntatori per impostare i puntatori in modo che puntino al di fuori dei confini dell'array è un errore logico che può portare a problemi di sicurezza.

7.8.5 Incrementare e decrementare un puntatore

Se un puntatore è incrementato o decrementato di uno, è possibile usare gli operatori di incremento (`++`) e decremento (`--`). Entrambe le istruzioni

```
++vPtr;  
vPtr++;
```

incrementano il puntatore facendolo puntare alla locazione successiva nell'array. Entrambe le istruzioni

```
--vPtr;  
vPtr--;
```

decrementano il puntatore facendolo puntare all'elemento precedente dell'array.

7.8.6 Sottrarre un puntatore da un altro

Se `vPtr` contiene la locazione 3000 e `v2Ptr` contiene l'indirizzo 3008, l'istruzione

```
x = v2Ptr - vPtr;
```

assegna a `x` il *numero degli elementi dell'array* tra `vPtr` e `v2Ptr`, in questo caso 2 (non 8). L'aritmetica dei puntatori è indefinita, a meno che non sia eseguita su elementi dello stesso array. Non possiamo presumere che due variabili dello stesso tipo siano memorizzate in modo contiguo nella memoria, a meno che non siano elementi adiacenti di un array.

7.8.7 Assegnare puntatori ad altri puntatori

Un puntatore può essere assegnato a un altro puntatore se entrambi hanno lo stesso tipo. L'eccezione a questa regola è costituita dal **puntatore a void** (cioè `void *`), il quale è un puntatore generico che può rappresentare *qualsiasi* tipo di puntatore. A tutti i tipi di puntatore è possibile assegnare un puntatore a `void *`, e a un `void *` è possibile assegnare un puntatore di qualsiasi tipo (compreso un altro `void *`). In entrambi i casi *non* è necessaria alcuna operazione di cast.

7.8.8 Puntatore a void

Un puntatore a `void` non può essere dereferenziato. Si può fare infatti la seguente considerazione: il compilatore sa che su una macchina con interi a 4 byte un `int *` punta a 4 byte di memoria. Tuttavia, un puntatore `void *` contiene una locazione di memoria per un tipo sconosciuto (il numero esatto di byte a cui il puntatore fa riferimento non è conosciuto dal compilatore). Il compilatore deve conoscere il tipo per determinare il numero di byte che rappresentano il valore di riferimento. Dereferenziare un puntatore `void *` è un errore di sintassi.

7.8.9 Confrontare i puntatori

È possibile confrontare i puntatori usando gli operatori di uguaglianza e relazionali, ma tali confronti hanno senso solo se i puntatori puntano a elementi dello stesso array; altrimenti, tali confronti sono errori logici. I confronti di puntatori comparano gli indirizzi memorizzati nei puntatori. Un confronto di questo tipo potrebbe evidenziare, per esempio, che uno dei due puntatori, rispetto all'altro, punta a un elemento dell'array con indice più alto. Il confronto di puntatori è comunemente usato per determinare se un puntatore è `NULL`.

✓ Autovalutazione

1. (*Completare*) Quando un intero è sommato a un puntatore o sottratto da esso, il puntatore è incrementato o decrementato di quell'intero moltiplicato per _____.

Risposta: la dimensione dell'oggetto a cui il puntatore fa riferimento.

2. (*Completare*) I puntatori `v1Ptr` e `v2Ptr` puntano a elementi dello stesso array di valori `double` di 8 byte. Se `v1Ptr` contiene l'indirizzo 3000 e `v2Ptr` contiene l'indirizzo 3016, allora l'istruzione

```
size_t x = v2Ptr - v1Ptr;
```

assegnerà _____ a `x`.

Risposta: 2 (non 16): 2 è il numero degli elementi tra i puntatori.

7.9 Relazioni tra puntatori e array

Array e puntatori sono intimamente correlati e spesso possono essere usati in maniera intercambiabile. Al nome di un array si può pensare come a un puntatore costante al primo elemento dell'array. I puntatori possono essere usati per fare qualsiasi operazione che implichi l'indicizzazione di un array.

Supponete le seguenti definizioni:

```
int b[5];
int *bPtr;
```

Poiché il nome dell'array `b` (senza indice) è un puntatore al primo elemento dell'array, possiamo impostare `bPtr` uguale all'indirizzo del primo elemento nell'array `b` con l'istruzione:

```
bPtr = b;
```

Questa istruzione è equivalente all'assegnazione dell'indirizzo del primo elemento dell'array `b` come segue:

```
bPtr = &b[0];
```

7.9.1 Notazione puntatore/offset

All'elemento `b[3]` dell'array si può fare alternativamente riferimento con l'espressione con puntatori

```
*(bPtr + 3)
```

Il 3 nell'espressione è l'**offset** (letteralmente "scarto") rispetto al puntatore. Quando `bPtr` punta al primo elemento di un array, l'offset indica a quale elemento dell'array si fa riferimento; il valore dell'offset coincide con l'indice dell'array. Questa notazione è detta **notazione puntatore/offset**. Le parentesi sono necessarie perché la precedenza dell'operatore `*` è maggiore della precedenza dell'operatore `+`. Senza le parentesi, l'espressione

precedente aggiungerebbe 3 al valore dell'espressione `*bPtr` (cioè il 3 verrebbe aggiunto a `b[0]`, supponendo che `bPtr` punti all'inizio dell'array). Così come si può fare riferimento all'elemento dell'array con un'espressione con puntatori, l'indirizzo

`&b[3]`

può essere scritto con la seguente espressione con puntatori

`bPtr + 3`

È possibile trattare lo stesso array come un puntatore e usarlo nell'aritmetica dei puntatori. Per esempio, l'espressione

`*(b + 3)`

fa riferimento all'elemento `b[3]`. In generale, è possibile scrivere con un puntatore e un offset tutte le espressioni con array indicizzati. In questo caso, la notazione puntatore/offset è stata usata con il nome dell'array come puntatore. L'istruzione precedente non modifica in alcun modo il nome dell'array; `b` punta ancora al primo elemento nell'array.

7.9.2 Notazione puntatore/indice

I puntatori possono essere indicizzati come gli array. Se `bPtr` ha il valore di `b`, l'espressione

`bPtr[1]`

si riferisce all'elemento dell'array `b[1]`. Questa è chiamata **notazione puntatore/indice**.

7.9.3 Non è possibile modificare il nome di un array con l'aritmetica dei puntatori

Il nome di un array punta sempre all'inizio dell'array, quindi è come un puntatore costante. Pertanto, l'espressione

`b += 3`

è scorretta poiché tenta di modificare il valore del nome dell'array con l'aritmetica dei puntatori. Tentare di modificare il valore del nome di un array con l'aritmetica dei puntatori genera un errore di compilazione.

7.9.4 Dimostrare indicizzazione e offset con un puntatore

Il programma della Figura 7.14 usa i quattro metodi che abbiamo esaminato per il riferimento agli elementi di un array (indicizzazione di un array, puntatore/offset con il nome dell'array come puntatore, **indicizzazione di un puntatore** e puntatore/offset con un puntatore) per stampare i quattro elementi dell'array intero `b`.

```

1 // fig07_14.cpp
2 // Uso delle notazioni con indice e con puntatori per gli array.
3 #include <stdio.h>
4 #define ARRAY_SIZE 4
5
6 int main(void) {
7     int b[] = {10, 20, 30, 40}; // crea e inizializza l'array b
8     int *bPtr = b; // crea bPtr e fallo puntare all'array b
9
10    // stampa l'array b usando la notazione degli array con indice
11    puts("Array b printed with:\nArray subscript notation");
12
13    // effettua un ciclo lungo l'array b
14    for (size_t i = 0; i < ARRAY_SIZE; ++i) {
15        printf("b[%zu] = %d\n", i, b[i]);

```

```

16 }
17
18 // stampa l'array b con il nome e la notazione puntatore/offset
19 puts("\nPointer/offset notation where the pointer is the array name");
20
21 // effettua un ciclo lungo l'array b
22 for (size_t offset = 0; offset < ARRAY_SIZE; ++offset) {
23     printf("*(%b + %zu) = %d\n", offset, *(b + offset));
24 }
25
26 // stampa l'array b con bPtr e la notazione degli array con indice
27 puts("\nPointer subscript notation");
28
29 // effettua un ciclo lungo l'array b
30 for (size_t i = 0; i < ARRAY_SIZE; ++i) {
31     printf("bPtr[%zu] = %d\n", i, bPtr[i]);
32 }
33
34 // stampa l'array b usando bPtr e la notazione puntatore/offset
35 puts("\nPointer/offset notation");
36
37 // effettua un ciclo lungo l'array b
38 for (size_t offset = 0; offset < ARRAY_SIZE; ++offset) {
39     printf("*(%bPtr + %zu) = %d\n", offset, *(bPtr + offset));
40 }
41 }
```

```

Array b printed with:
Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

Pointer/offset notation where the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

Figura 7.14 Uso delle notazioni con indice e con puntatori per gli array.

7.9.5 Copiare stringhe con array e puntatori

Per illustrare ulteriormente l'intercambiabilità fra array e puntatori, esaminiamo le due funzioni che copiano stringhe, `copy1` e `copy2`, nel programma della Figura 7.15. Entrambe le funzioni copiano una stringa in un array di caratteri, ma sono implementate in maniera diversa.

```

1 // fig07_15.c
2 // Copia di una stringa usando le notazioni con array e con puntatori.
3 #include <stdio.h>
4 #define SIZE 10
5
6 void copy1(char * const s1, const char * const s2); // prototipo
7 void copy2(char *s1, const char *s2); // prototipo
8
9 int main(void) {
10     char string1[SIZE]; // crea l'array string1
11     char *string2 = "Hello"; // crea un puntatore a una stringa
12
13     copy1(string1, string2);
14     printf("string1 = %s\n", string1);
15
16     char string3[SIZE]; // crea l'array string3
17     char string4[] = "Good Bye"; // crea un array contenente una stringa
18
19     copy2(string3, string4);
20     printf("string3 = %s\n", string3);
21 }
22
23 // copia s2 in s1 usando la notazione con array
24 void copy1(char * const s1, const char * const s2) {
25     // effettua un ciclo lungo le stringhe
26     for (size_t i = 0; (s1[i] = s2[i]) != ; ++i) {
27         ; // non fare niente nel corpo
28     }
29 }
30
31 // copia s2 in s1 usando la notazione con puntatori
32 void copy2(char *s1, const char *s2) {
33     // effettua un ciclo lungo le stringhe
34     for (; (*s1 = *s2) != ; ++s1, ++s2) {
35         ; // non fare niente nel corpo
36     }
37 }
```

```

string1 = Hello
string3 = Good Bye
```

Figura 7.15 Copia di una stringa usando le notazioni con array e con puntatori.

Copiare con la notazione degli array con indice

La funzione `copy1` usa la *notazione degli array con indice* per copiare la stringa in `s2` nell'array di caratteri `s1`. La funzione definisce la variabile contatore `i` come indice dell'array. L'intestazione dell'istruzione `for` (riga 26) esegue l'intera operazione di copiatura. Il corpo dell'istruzione è l'istruzione vuota. L'intestazione specifica che `i` è inizializzata a zero e viene incrementata di uno a ogni iterazione del ciclo. L'espressione `s1[i] = s2[i]` copia un carattere da `s2` a `s1`. Quando in `s2` si incontra il carattere nullo, questo viene assegnato a `s1`. Poiché il valore dell'assegnazione diventa il valore assegnato all'operando sinistro (`s1`), il ciclo termina quando un elemento di `s1` riceve il carattere nullo, che ha il valore 0 e quindi è *falso*.

Copiare con puntatori e aritmetica dei puntatori

La funzione `copy2` usa *puntatori e l'aritmetica dei puntatori* per copiare la stringa in `s2` nell'array di caratteri `s1`. Ancora, l'intestazione dell'istruzione `for` (riga 34) esegue l'intera operazione di copiatura. L'intestazione non include alcuna inizializzazione della variabile. L'espressione `*s1 = *s2` esegue l'operazione di copiatura dereferenziando `s2` e assegnando quel carattere alla locazione corrente in `s1`. Dopo l'assegnazione, la riga 34 incrementa `s1` e `s2` per puntare al carattere successivo di ciascuna stringa. Quando l'assegnazione copia il carattere nullo in `s1`, il ciclo termina.

Note riguardanti le funzioni `copy1` e `copy2`

 Il primo argomento sia per `copy1` che per `copy2` deve essere un array grande abbastanza da contenere la stringa fornita come secondo argomento. Altrimenti, può verificarsi un errore in seguito al tentativo di scrivere in una locazione di memoria che non fa parte dell'array. In entrambe le funzioni, il secondo argomento è copiato nel primo argomento – i caratteri sono letti da esso uno alla volta, ma non vengono mai modificati. Pertanto, il secondo parametro è dichiarato come puntatore a un valore costante in modo che venga applicato il principio del privilegio minimo. Nessuna delle due funzioni richiede la capacità di modifica della stringa nel secondo argomento, quindi semplicemente non la consideriamo.

✓ Autovalutazione

1. (*Vero/Falso*) Se `bPtr` punta al secondo elemento dell'array `b` (`b[1]`), si può fare riferimento all'elemento `b[3]` anche usando l'espressione con la notazione puntatore/offset `(bPtr + 3)`.

Risposta: Falso. Poiché il puntatore punta al *secondo elemento* dell'array `b` (`b[1]`), l'espressione deve essere `(bPtr + 2)`.

2. (*Completare*) I puntatori possono essere indicizzati come gli array. Se `bPtr` punta al primo elemento dell'array `b`, l'espressione

`bPtr[1]`

si riferisce all'elemento _____ dell'array.

Risposta: `b[1]`.

7.10 Array di puntatori

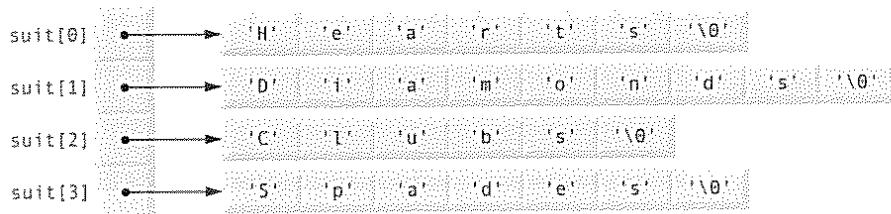
Gli array possono contenere puntatori. Un uso comune di un **array di puntatori** è quello di realizzare un **array di stringhe**. Ogni elemento in una stringa in C è sostanzialmente un puntatore al suo primo carattere. Pertanto ogni elemento in un array di stringhe è in realtà un puntatore al primo carattere di una stringa. Considerate la definizione dell'array di stringhe `suit`, che potrebbe essere utile per rappresentare un mazzo di carte da gioco.

```
const char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades"};
```

L'array ha 4 elementi. Il `char *` indica che ogni elemento di `suit` è del tipo “puntatore a `char`”. Il qualificatore `const` indica che la stringa a cui punta ciascun elemento non può essere modificata. Le stringhe “Hearts”, “Diamonds”, “Clubs” e “Spades” sono inserite nell'array. Ognuna è memorizzata come *stringa di caratteri*.

terminata da un carattere nullo, che è di un carattere più lunga del numero dei caratteri tra le virgolette. Quindi, le stringhe sono lunghe, rispettivamente, 7, 9, 6 e 7 caratteri.

Sebbene sembri che queste stringhe vengano inserite nell'array, di fatto solo i puntatori sono memorizzati, come mostrato nel seguente diagramma:



Ogni puntatore punta al primo carattere della stringa corrispondente. Così, anche se un array `char *` è fisso come dimensione, fornisce l'accesso a stringhe di caratteri di *qualsiasi lunghezza*. Questa flessibilità è un esempio delle potenti capacità di strutturazione dei dati del linguaggio C.

I semi delle carte da gioco avrebbero potuto essere collocati in un array bidimensionale, nel quale ogni riga avrebbe rappresentato un seme e ogni colonna una lettera del nome di un seme. Una tale struttura di dati avrebbe un numero fisso di colonne per riga e quel numero dovrebbe essere grande tanto quanto la stringa più grande. Potrebbe dunque andare sprecata una memoria considerevole quando si memorizza un grande numero di stringhe che sono più corte della stringa più lunga. Qui usiamo array di stringhe per rappresentare un mazzo di carte.

✓ Autovalutazione

- (Completare)* Un uso comune di un array di puntatori è quello di realizzare un array di _____.

Risposta: stringhe.

- (Vero/Falso)* I caratteri delle stringhe dell'array `suit` di questo paragrafo sono memorizzati direttamente negli elementi dell'array.

Risposta: Falso. Sebbene sembri che l'array contenga quattro stringhe, ciascun elemento contiene in realtà l'indirizzo del primo carattere della stringa corrispondente. Le lettere effettive e i caratteri nulli di terminazione sono memorizzati altrove in memoria.

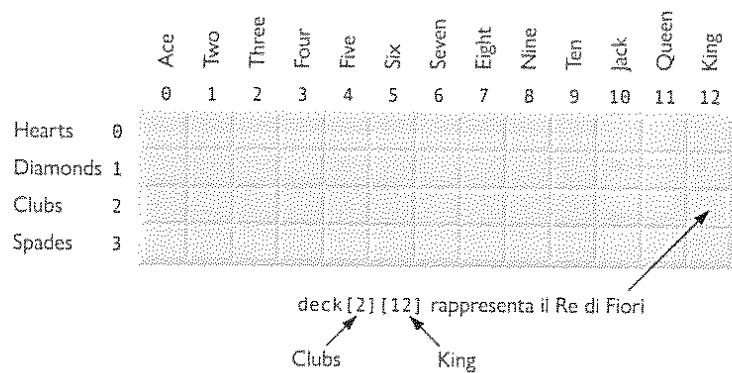
7.11 Caso pratico di simulazione di numeri casuali: mescolare e distribuire le carte

Useremo la generazione di numeri casuali per sviluppare un programma di simulazione di mescolamento e distribuzione delle carte, che può quindi essere utilizzato per implementare programmi per specifici giochi di carte. Per evidenziare alcuni sottili problemi di prestazioni, abbiamo intenzionalmente usato algoritmi subottimali per mescolare e distribuire le carte. Negli esercizi di questo capitolo e nel Capitolo 10 svilupperemo algoritmi più efficienti.

Usando l'approccio top-down con affinamento graduale, svilupperemo un programma che mescola un mazzo di 52 carte da gioco e poi le distribuisce. L'approccio top-down è particolarmente utile per affrontare problemi più grandi e più complessi di quelli che avete visto nei precedenti capitoli.

Rappresentare un mazzo di carte come array bidimensionale

Usiamo l'array `deck` bidimensionale 4-per-13 per rappresentare il mazzo di carte da gioco:



Le righe corrispondono ai *semi*: la riga 0 corrisponde ai Cuori (Hearts), la riga 1 ai Quadri (Diamonds), la riga 2 ai Fiori (Clubs) e la riga 3 alle Picche (Spades). Le colonne corrispondono ai *valori* delle carte. Le colonne da 0 a 9 corrispondono, rispettivamente, ai valori dall'asso al dieci e le colonne da 10 a 12 corrispondono al Fante, alla Regina e al Re. Caricheremo l'array di stringhe `suit` con le stringhe di caratteri che rappresentano i quattro semi, e l'array di stringhe `face` con stringhe che rappresentano i tredici valori delle carte.

Mescolare l'array bidimensionale

Questo mazzo di carte simulato può essere *mescolato* come segue. Dapprima, impostate tutti gli elementi del mazzo (`deck`) a 0. Poi, scegliete una riga (`row`) da 0 a 3 e una colonna (`column`) da 0 a 12 *a caso*. Inserite il numero 1 nell'elemento dell'array `deck[row][column]` per indicare che questa carta sarà la prima del mazzo mescolato a essere distribuita. Ripetete questo processo per i numeri 2, 3, ..., 52, inserendo ciascuno a caso nell'array `deck` per indicare quali carte sono da porre come seconda, terza, ... e cinquantaduesima nel mazzo mescolato. Quando l'array `deck` comincia a essere pieno di numeri di carte, è possibile che una carta venga selezionata di nuovo, cioè, quando verrà selezionato, `deck[row][column]` risulterà già diverso da zero. Ignorate questa selezione e scegliete ripetutamente a caso altre righe (`row`) e colonne (`column`) finché non verrà trovata una carta *non ancora selezionata*. Alla fine, i numeri da 1 a 52 occuperanno i 52 spazi dell'array `deck`. A questo punto il mazzo di carte è completamente mescolato.

Possibilità di posposizione indefinita

Questo algoritmo per mescolare le carte può venire eseguito *indefinitamente* se le carte che sono già state mescolate vengono ripetutamente selezionate a caso. Questo fenomeno è noto come **posposizione indefinita**. Negli esercizi di questo capitolo esamineremo un algoritmo migliore per mescolare le carte, che elimina la possibilità della posposizione indefinita.

A volte un algoritmo che viene individuato in un modo "naturale" può presentare sottili problemi di prestazioni, come la posposizione indefinita. Cercate algoritmi che la evitino.

Distribuire le carte dell'array bidimensionale

Per distribuire la prima carta cerchiamo nell'array l'elemento `deck[row][column]` uguale a 1 usando istruzioni `for` annidate che fanno variare `row` da 0 a 3 e `column` da 0 a 12. A quale carta corrisponde quell'elemento dell'array? L'array `suit` è stato precaricato con i quattro semi, per cui, per ottenere il seme, stampiamo la stringa di caratteri `suit[row]`. In modo simile, per ottenere il valore della carta, stampiamo la stringa di caratteri `face[column]`. Stampiamo anche la stringa di caratteri " of ", come in "King of Clubs", "Ace of Diamonds" e così via.

Sviluppare la logica del programma con il processo top-down di affinamento graduale
Procediamo con il processo top-down di affinamento graduale. Il *top* è semplicemente:

Mescola e distribuisci le 52 carte

Il nostro *primo affinamento* produce:

Inizializza l'array dei semi

Inizializza l'array dei valori

Inizializza l'array del mazzo

Mescola il mazzo

Distribuisci le 52 carte

“Mescola il mazzo” può essere affinato come segue:

Per ognuna delle 52 carte

Inserisci il numero della carta in un elemento non occupato selezionato a caso dell'array del mazzo

“Distribuisci le 52 carte” può essere affinato come segue:

Per ognuna delle 52 carte

Trova il numero della carta nell'array del mazzo e stampa il valore e il seme della carta

Il *secondo affinamento* completo è:

Inizializza l'array dei semi

Inizializza l'array dei valori

Inizializza l'array del mazzo

Per ognuna delle 52 carte

Inserisci il numero della carta in un elemento non occupato selezionato a caso dell'array del mazzo

Per ognuna delle 52 carte

Trova il numero della carta nell'array del mazzo e stampa il valore e il seme della carta

“Inserisci il numero della carta in un elemento non occupato selezionato a caso del mazzo” può essere affinato come:

Scegli a caso un elemento dell'array del mazzo

Finché l'elemento scelto dell'array del mazzo risulta già precedentemente scelto

 Seegli a caso un elemento dell'array del mazzo

 Inserisci il numero della carta nell'elemento scelto dell'array del mazzo

“Trova il numero della carta nell'array del mazzo e stampa il valore e il seme della carta” può essere affinato come:

Per ogni elemento dell'array del mazzo

 Se l'elemento contiene il numero della carta

 Stampa il valore e il seme della carta

Accorpando queste espansioni si ottiene il nostro *terzo affinamento*:

Inizializza l'array dei semi

Inizializza l'array dei valori

Inizializza l'array del mazzo

Per ognuna delle 52 carte

 Scegli a caso un elemento dell'array del mazzo

Finché l'elemento scelto dell'array del mazzo risulta già precedentemente scelto
 Scegli a caso un elemento dell'array del mazzo

Inserisci il numero della carta nell'elemento scelto dell'array del mazzo

Per ognuna delle 52 carte

Per ogni elemento dell'array del mazzo

Se l'elemento contiene il numero di carta desiderato

Stampa il valore e il seme della carta

Questo completa il processo di affinamento.

Implementare il programma che mescola e distribuisce le carte

Il programma che mescola e distribuisce le carte e un'esecuzione di esempio sono mostrati nella Figura 7.16. Quando la funzione `printf` usa la specifica di conversione `%s` per stampare una stringa, l'argomento corrispondente deve essere un puntatore a `char` che punta a una stringa o un array `char` che contiene una stringa. La specifica di formato della riga 59 stampa il valore della carta *allineato a destra* in un campo di cinque caratteri seguito da "of" e il seme della carta *allineato a sinistra* in un campo di otto caratteri. Il *segno meno* in `%-8s` indica allineamento a sinistra.

```

1 // fig07_16.c
2 // Mescolare e distribuire le carte.
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 #define SUITS 4
8 #define FACES 13
9 #define CARDS 52
10
11 // prototipi
12 void shuffle(int deck[][FACES]);
13 void deal(int deck[][FACES], const char *face[], const char *suit[]);
14
15 int main(void) {
16     // inizializza l'array deck
17     int deck[SUITS][FACES] = {0};
18
19     srand(time(NULL)); // seme per i numeri casuali
20     shuffle(deck); // mescola il mazzo
21
22     // inizializza l'array suit
23     const char *suit[SUITS] = {"Hearts", "Diamonds", "Clubs", "Spades"};
24
25     // inizializza l'array face
26     const char *face[FACES] = {"Ace", "Deuce", "Three", "Four", "Five",
27         "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King"};
28
29     deal(deck, face, suit); // distribuisci il mazzo
30 }
31
32 // mescola le carte nel mazzo
33 void shuffle(int deck[][FACES]) {
34     // per ognuna delle carte, scegli un elemento del mazzo a caso
35     for (size_t card = 1; card <= CARDS; ++card) {

```

```

36     size_t row = 0; // numero della riga
37     size_t column = 0; // numero della colonna
38
39     // scegli una nuova locazione a caso non occupata
40     do {
41         row = rand() % SUITS;
42         column = rand() % FACES;
43     } while(deck[row][column] != 0);
44
45     deck[row][column] = card; // poni numero della carta in elemento scelto
46 }
47 }
48
49 // distribuisci le carte nel mazzo
50 void deal(int deck[][FACES], const char *face[], const char *suit[]) {
51     // distribuisci ognuna delle carte
52     for (size_t card = 1; card <= CARDS; ++card) {
53         // effettua un ciclo lungo le righe di deck
54         for (size_t row = 0; row < SUITS; ++row) {
55             // effettua un ciclo lungo le colonne di deck per la riga corrente
56             for (size_t column = 0; column < FACES; ++column) {
57                 // se l'elemento contiene la carta corrente, stampala
58                 if (deck[row][column] == card) {
59                     printf("%5s of %-8s %c", face[column], suit[row],
60                           card % 4 == 0 ? : ); // formato a 2 colonne
61                 }
62             }
63         }
64     }
65 }
```

Ace of Hearts	Jack of Hearts	Five of Clubs	King of Clubs
Eight of Diamonds	Three of Clubs	Deuce of Hearts	Four of Hearts
Ace of Clubs	Deuce of Spades	Queen of Diamonds	Six of Hearts
Seven of Clubs	Five of Hearts	Deuce of Clubs	King of Hearts
Nine of Spades	Ace of Spades	Ace of Diamonds	Eight of Spades
Eight of Hearts	Ten of Spades	Ten of Hearts	Queen of Clubs
Jack of Spades	Jack of Diamonds	Three of Spades	Four of Clubs
Four of Spades	Ten of Clubs	King of Diamonds	Six of Spades
Nine of Clubs	Six of Diamonds	Queen of Spades	King of Spades
Four of Diamonds	Eight of Clubs	Jack of Clubs	Seven of Hearts
Seven of Diamonds	Three of Hearts	Five of Spades	Nine of Hearts
Nine of Diamonds	Three of Diamonds	Deuce of Diamonds	Queen of Hearts
Six of Clubs	Seven of Spades	Five of Diamonds	Ten of Diamonds

Figura 7.16 Mescolare e distribuire le carte.**Perfezionare l'algoritmo per distribuire le carte**

C'è un punto di debolezza nell'algoritmo per distribuire le carte. Una volta che viene localizzata la carta cercata, le due istruzioni `for` interne continuano ancora la ricerca nei restanti elementi del mazzo. Correggeremo tale difetto negli esercizi di questo capitolo e in un caso pratico del Capitolo 10.

Esercizi correlati

Questo caso pratico per mescolare e distribuire le carte è affiancato dai seguenti esercizi.

- Esercizio 7.12 (Mescolare e distribuire le carte: distribuire mani di poker).
- Esercizio 7.13 (Progetto: mescolare e distribuire le carte – Qual è la mano di poker migliore?).
- Esercizio 7.14 (Progetto: mescolare e distribuire le carte – Simulare la persona che distribuisce le carte).
- Esercizio 7.15 (Progetto: mescolare e distribuire le carte – Consentire ai giocatori di pescare carte).
- Esercizio 7.16 (Modifica al programma per mescolare e distribuire le carte: mescolamento a elevate prestazioni).

✓ Autovalutazione

1. (*Completare*) L'algoritmo per mescolare le carte che abbiamo presentato può essere eseguito indefinitamente se le carte che sono già state mescolate vengono ripetutamente selezionate a caso. Questo fenomeno è noto come _____.

Risposta: posposizione indefinita.

2. (*Vero/Falso*) La specifica di formato "%5s of %-8s" stampa una stringa *allineata a sinistra* in un campo di cinque caratteri seguita da "of" e da una stringa *allineata a destra* in un campo di otto caratteri.

Risposta: *Falso*. Questa specifica di formato stampa una stringa *allineata a destra* in un campo di cinque caratteri seguita da "of" e da una stringa *allineata a sinistra* in un campo di otto caratteri.

7.12 Puntatori a funzioni

Nel Capitolo 6 abbiamo visto che il nome di un array è in realtà l'indirizzo in memoria del primo elemento dell'array. In modo simile, il nome di una funzione è in realtà l'indirizzo in memoria di partenza del codice che esegue il compito della funzione. I puntatori a funzioni possono essere passati alle funzioni, restituiti dalle funzioni, memorizzati negli array, assegnati ad altri puntatori a funzioni e confrontati tra loro per egualianza o inegualianza.

7.12.1 Ordinamento crescente o decrescente

Per illustrare l'uso dei puntatori a funzioni, la Figura 7.17 presenta una versione modificata del programma per il bubble sort della Figura 7.11. La nuova versione consiste nella funzione main e nelle funzioni bubbleSort, swap, ascending e descending. La funzione bubbleSort riceve come argomento un puntatore a una funzione – o alla funzione ascending o alla funzione descending – in aggiunta a un array di interi e alla dimensione dell'array. L'utente sceglie se ordinare l'array in ordine *crescente* (1) o *decrescente* (2). Se l'utente inserisce 1, main passa un puntatore alla funzione ascending alla funzione bubbleSort. Se l'utente inserisce 2, main passa un puntatore alla funzione descending alla funzione bubbleSort.

```

1 // fig07_17.c
2 // Programma multifunzione di ordinamento che usa puntatori a funzioni.
3 #include <stdio.h>
4 #define SIZE 10
5
6 // prototipi
7 void bubbleSort(int work[], size_t size, int (*compare)(int a, int b));
8 int ascending(int a, int b);
9 int descending(int a, int b);
10
11 int main(void) {
12     // inizializza l'array a non ordinato
13     int a[SIZE] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

```

```
14
15 printf("%s", "Enter 1 to sort in ascending order,\n"
16     "Enter 2 to sort in descending order: ");
17 int order = 0;
18 scanf("%d", &order);
19
20 puts("\nData items in original order");
21
22 // stampa l'array originario
23 for (size_t counter = 0; counter < SIZE; ++counter) {
24     printf("%5d", a[counter]);
25 }
26
27 // ordina l'array in ordine crescente; passa la funzione ascending
28 // come argomento per specificare l'ordine crescente dell'ordinamento
29 if (order == 1) {
30     bubbleSort(a, SIZE, ascending);
31     puts("\nData items in ascending order");
32 }
33 else { // passa la funzione descending
34     bubbleSort(a, SIZE, descending);
35     puts("\nData items in descending order");
36 }
37
38 // stampa l'array ordinato
39 for (size_t counter = 0; counter < SIZE; ++counter) {
40     printf("%5d", a[counter]);
41 }
42
43 puts("\n");
44 }
45
46 // bubble sort multifunzione; il parametro compare e' un puntatore
47 // alla funzione di confronto che determina il tipo di ordinamento
48 void bubbleSort(int work[], size_t size, int (*compare)(int a, int b)) {
49     void swap(int *element1Ptr, int *element2ptr); // prototipo
50
51     // ciclo di controllo delle passate
52     for (int pass = 1; pass < size; ++pass) {
53         // ciclo di controllo del numero di confronti per passata
54         for (size_t count = 0; count < size - 1; ++count) {
55             // se elementi adiacenti non sono in ordine, scambiali
56             if ((*compare)(work[count], work[count + 1])) {
57                 swap(&work[count], &work[count + 1]);
58             }
59         }
60     }
61 }
62
63 // scambia i valori alle locazioni di memoria a cui puntano
64 // element1Ptr ed element2Ptr
65 void swap(int *element1Ptr, int *element2Ptr) {
66     int hold = *element1Ptr;
```

```

67     *element1Ptr = *element2Ptr;
68     *element2Ptr = hold;
69 }
70
71 // determina se gli elementi non sono in ordine per un ordinamento crescente
72 int ascending(int a, int b) {
73     return b < a; // si deve effettuare lo scambio se b e' minore di a
74 }
75
76 // determina se gli elementi non sono in ordine per un ordinamento decrescente
77 int descending(int a, int b) {
78     return b > a; // si deve effettuare lo scambio se b e' maggiore di a
79 }

```

```

Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1

Data items in original order
2 6 4 8 10 12 89 68 45 37
Data items in ascending order
2 4 6 8 10 12 37 45 68 89

```

```

Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2

Data items in original order
2 6 4 8 10 12 89 68 45 37
Data items in descending order
89 68 45 37 12 10 8 6 4 2

```

Figura 7.17 Programma multifunzione di ordinamento che usa puntatori a funzioni.

Parametro per puntatore a funzione

Il seguente parametro è dichiarato nell'intestazione della funzione per bubbleSort (riga 48):

```
int (*compare)(int a, int b)
```

Questa dichiarazione dice a bubbleSort che deve aspettarsi un parametro (`compare`) che è un puntatore a una funzione, specificamente per una funzione che riceve due `int` e restituisce un risultato `int`. Sono necessarie le parentesi attorno a `*compare` per associare `*` a `compare` e indicare che `compare` è un *puntatore*. Senza le parentesi, la dichiarazione sarebbe stata

```
int *compare(int a, int b)
```

che dichiara una funzione che riceve due interi come parametri e restituisce un puntatore a un intero.

Per chiamare la funzione passata a bubbleSort tramite il suo puntatore a funzione, lo dereferenziamo, come mostrato nell'istruzione `if` alla riga 56:

```
if ((*compare)(work[count], work[count + 1]))
```

La chiamata alla funzione avrebbe potuto essere fatta senza dereferenziare il puntatore come in

```
if (compare(work[count], work[count + 1]))
```

che usa il puntatore direttamente come nome della funzione. Il primo metodo per chiamare una funzione per mezzo di un puntatore fa vedere esplicitamente che `compare` è un puntatore a una funzione che è dereferen-

ziato per chiamare la funzione. Il secondo metodo fa sì che compare sembri il nome di una funzione *vera*. Ciò può essere fonte di confusione per un programmatore che legge il codice e si aspetterebbe una definizione della funzione compare mentre essa non è mai definita.

7.12.2 Uso dei puntatori a funzioni per realizzare un sistema guidato da menu

Un uso comune dei **puntatori a funzioni** si ha nei *sistemi guidati da menu*. Un programma richiede a un utente di selezionare un'opzione da un menu (per esempio, da 0 a 2) scrivendo il numero dell'elemento del menu. Il programma realizza ciascuna opzione con una diversa funzione e memorizza i puntatori a ciascuna funzione in un array di puntatori a funzioni. La scelta dell'utente è usata come indice dell'array e il puntatore nell'array è usato per chiamare la funzione.

Il programma della Figura 7.18 fornisce un esempio generico dei meccanismi utilizzati per definire e usare un array di puntatori a funzioni. Definiamo tre funzioni – function1, function2 e function3 – che ricevono ciascuna un argomento intero e non restituiscono niente. Memorizziamo i puntatori a queste tre funzioni nell'array f (riga 13). La definizione si legge iniziando dall'insieme di parentesi più a sinistra, “f è un array di 3 puntatori a funzioni che ricevono ognuna un int come argomento e restituiscono void”. L'array è inizializzato con i nomi delle tre funzioni. Quando l'utente inserisce un valore tra 0 e 2, il valore viene usato come indice nell'array dei puntatori alle funzioni. Nella chiamata di funzione (riga 23), f[choice] seleziona il puntatore nella locazione choice nell'array. *Dereferenziamo il puntatore per chiamare la funzione*, passando choice come argomento della funzione. Ogni funzione stampa il valore del suo argomento e il suo nome per confermare che la funzione è stata chiamata correttamente. Negli esercizi di questo capitolo svilupperete diversi sistemi guidati da menu.

```

1 // fig07_18.c
2 // Esempio di un array di puntatori a funzioni.
3 #include <stdio.h>
4
5 // prototipi
6 void function1(int a);
7 void function2(int b);
8 void function3(int c);
9
10 int main(void) {
11     // inizializza un array di 3 puntatori a funzioni che ricevono
12     // ognuna un argomento int e restituiscono void
13     void (*f[3])(int) = {function1, function2, function3};
14
15     printf("%s", "Enter a number between 0 and 2, 3 to end: ");
16     int choice = 0;
17     scanf("%d", &choice);
18
19     // elabora la scelta dell'utente
20     while (choice >= 0 && choice < 3) {
21         // invoca la funzione alla locazione choice nell'array f e passa
22         // choice come argomento
23         (*f[choice])(choice);
24
25         printf("%s", "Enter a number between 0 and 2, 3 to end: ");
26         scanf("%d", &choice);
27     }
28
29     puts("Program execution completed.");
30 }
31

```

```

32 void function1(int a) {
33     printf("You entered %d so function1 was called\n\n", a);
34 }
35
36 void function2(int b) {
37     printf("You entered %d so function2 was called\n\n", b);
38 }
39
40 void Function3(int c) {
41     printf("You entered %d so function3 was called\n\n", c);
42 }

```

```
Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function1 was called
```

```
Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function2 was called
```

```
Enter a number between 0 and 2, 3 to end: 2
You entered 2 so function3 was called
```

```
Enter a number between 0 and 2, 3 to end: 3
Program execution completed.
```

Figura 7.18 Esempio di un array di puntatori a funzioni.

✓ Autovalutazione

1. (*Vero/Falso*) Considerate il parametro seguente che è stato dichiarato nell'intestazione di funzione per la funzione bubble sort del nostro esempio:

```
int (*compare)(int a, int b)
```

Questo parametro *compare* è un puntatore a una funzione che riceve due parametri interi e restituisce un risultato intero. Le parentesi attorno a **compare* sono opzionali ma le includiamo per maggiore chiarezza.

Risposta: *Falso*. Le parentesi sono necessarie per raggruppare **compare* e indicare che *compare* è un puntatore. Senza le parentesi, la dichiarazione sarebbe stata

```
int *compare(int a, int b)
```

che è semplicemente l'intestazione di una funzione che riceve due interi come parametri e *restituisce un puntatore a un intero*.

2. (*Completare*) Proprio come un puntatore a una variabile è dereferenziato per accedere al valore della variabile, un puntatore a una funzione è dereferenziato per _____.

Risposta: chiamare la funzione.

7.13 Programmazione sicura in C

printf_s, scanf_s e altre funzioni sicure

I precedenti paragrafi sulla programmazione sicura in C hanno presentato *printf_s* e *scanf_s* e hanno menzionato altre versioni più sicure delle funzioni della Libreria Standard descritte dall'Annex K del C standard. Una caratteristica chiave di funzioni come *printf_s* e *scanf_s* che le rende più sicure è quella di avere *restrizioni in fase di esecuzione* che richiedono che i loro argomenti puntatore siano non-NUL. Le funzioni

controllano queste restrizioni in fase di esecuzione *prima* di provare a usare i puntatori. Qualunque argomento che sia un puntatore NULL è una *violazione alla restrizione*, fa fallire la funzione e le fa restituire una notifica del suo stato. Una chiamata alla funzione `scanf_s` restituisce EOF se uno qualsiasi degli argomenti puntatore (compresa la stringa di controllo del formato) è NULL. Una chiamata alla funzione `printf_s` arresta la stampa dei dati e restituisce un numero negativo se la stringa di controllo del formato è NULL o un argomento che corrisponde a un %s è NULL. Per dettagli completi delle funzioni dell'Annex K, consultate il documento del C standard o la documentazione della libreria del vostro compilatore.

Altre linee guida del CERT riguardanti i puntatori

L'uso scorretto dei puntatori porta oggi a molte delle più comuni vulnerabilità nella sicurezza dei sistemi. Il CERT fornisce varie linee guida per aiutare a evitare tali problemi. Se avete intenzione di costruire sistemi in C a livello industriale, dovete familiarizzare con il *CERT C Secure Coding Standard* all'indirizzo <https://wiki.sei.cmu.edu/>. Le seguenti linee guida si applicano alle tecniche di programmazione con puntatori che abbiamo presentato in questo capitolo:

- EXP34-C: dereferenziare i puntatori NULL causa tipicamente l'arresto anomalo dei programmi, ma il CERT ha individuato situazioni in cui dereferenziare i puntatori NULL può permettere agli autori di un attacco di far eseguire del codice.
- DCL13-C: il Paragrafo 7.5 ha esaminato l'uso di `const` con i puntatori. Se il parametro di una funzione punta a un valore che non verrà cambiato dalla funzione, si deve usare `const` per indicare che i dati sono costanti. Per esempio, per rappresentare un puntatore a una stringa che non verrà modificata, usate `const char *` come tipo del parametro puntatore.
- WIN04-C: questa linea guida esamina le tecniche per criptare i puntatori a funzioni su Windows di Microsoft, in modo da evitare che autori di attacchi li sovrascrivano ed eseguano il loro codice.

✓ Autovalutazione

1. (*Completare*) Una caratteristica chiave di funzioni come `printf_s` e `scanf_s` è quella di avere restrizioni in fase di esecuzione che richiedono che i loro argomenti puntatore siano _____.

Risposta: non-NUL.

2. (*Vero/Falso*) L'uso scorretto dei puntatori porta oggi a molte delle più comuni vulnerabilità nella sicurezza dei sistemi.

Risposta: Vero.

7.14 Riepilogo

Paragrafo 7.2 Definizioni e inizializzazione di variabili puntatore

- Un puntatore contiene un indirizzo di un'altra variabile che contiene un valore. In questo senso, il nome di una variabile fa riferimento *direttamente* a un valore e un puntatore fa riferimento *indirettamente* a un valore.
- Fare riferimento a un valore per mezzo di un puntatore si dice **indirezione**.
- I puntatori possono essere definiti per puntare a oggetti di qualsiasi tipo.
- I puntatori vanno inizializzati o quando sono definiti o in un'istruzione di assegnazione. Un puntatore si può inizializzare a NULL, 0 oppure a un indirizzo. Un puntatore con il valore NULL non punta a niente. Inizializzare un puntatore a 0 equivale a inizializzarlo a NULL, ma NULL è preferibile per chiarezza. Il valore 0 è l'unico valore intero che può essere assegnato direttamente a una variabile puntatore.
- NULL è una costante simbolica definita nel file di intestazione `<stddef.h>` (e in diversi altri file di intestazione).

Paragrafo 7.3 Operatori per i puntatori

- L'operatore di indirizzo `&` è un operatore unario che restituisce l'indirizzo del suo operando.
- L'operando dell'operatore di indirizzo deve essere una variabile.
- L'operatore di indirezione `*` restituisce il valore dell'oggetto al quale punta il suo operando.
- La specifica di conversione `%p` di `printf` stampa un indirizzo di memoria rappresentato con un intero esadecimale su gran parte delle piattaforme.

Paragrafo 7.4 Passare argomenti a funzioni per riferimento

- In C, gli argomenti (a differenza degli array) sono passati per valore.
- I programmi in C compiono il passaggio per riferimento usando i puntatori e l'operatore di indirezione. Per passare una variabile per riferimento, applicate l'operatore di indirizzo (`&`) al nome della variabile.
- Quando l'indirizzo di una variabile viene passato a una funzione, è possibile usare l'operatore di indirezione (`*`) nella funzione per leggere e/o modificare il valore in quella locazione nella memoria della funzione chiamante.
- Una funzione che riceve un indirizzo come argomento deve definire un parametro puntatore per ricevere l'indirizzo.
- Il compilatore non distingue fra una funzione che riceve un puntatore e una che riceve un array unidimensionale. Una funzione deve “sapere” quando sta per ricevere un array invece di una singola variabile passata per riferimento.
- Quando il compilatore incontra il parametro di una funzione per un array unidimensionale della forma `int b[]` lo converte nella notazione con puntatore `int *b`.

Paragrafo 7.5 Uso del qualificatore `const` con i puntatori

- Il qualificatore `const` indica che il valore di una particolare variabile non va modificato.
- Vi sono quattro modi di passare un puntatore a una funzione: un puntatore non costante a dati non costanti, un puntatore costante a dati non costanti, un puntatore non costante a dati costanti e un puntatore costante a dati costanti.
- Con un puntatore non costante a dati non costanti, è possibile modificare i dati per mezzo del puntatore dereferenziato e modificare il puntatore per puntare ad altri dati.
- È possibile modificare un puntatore non costante a dati costanti per puntare a un qualunque dato di tipo appropriato, ma non è possibile modificare i dati a cui punta.
- Un puntatore costante a dati non costanti punta sempre alla stessa locazione di memoria ed è possibile modificare i dati in quella locazione tramite il puntatore. Questa è l'impostazione predefinita per il nome di un array.
- Un puntatore costante a dati costanti punta sempre alla stessa locazione di memoria e non è possibile modificare i dati in quella locazione di memoria.

Paragrafo 7.7 Operatore `sizeof`

- L'operatore unario `sizeof` determina la dimensione in byte di una variabile o di un tipo.
- Quando è applicato al nome di un array, `sizeof` restituisce il numero totale di byte nell'array.
- È possibile applicare l'operatore `sizeof` al nome, al tipo o al valore di una qualsiasi variabile.
- È necessario usare le parentesi con `sizeof` quando il suo operando è un tipo di dati.

Paragrafo 7.8 Espressioni con puntatori e aritmetica dei puntatori

- È possibile eseguire sui puntatori un insieme limitato di operazioni aritmetiche. È possibile incrementare (`++`) o decrementare (`--`) un puntatore, sommare un intero a un puntatore (`+ o +=`), sottrarre un intero da un puntatore (`- o -=`) e sottrarre un puntatore da un altro.

- Quando un intero è sommato a un puntatore o sottratto da esso, il puntatore è incrementato o decrementato di una quantità pari a quell'intero moltiplicato per le dimensioni dell'oggetto a cui il puntatore si riferisce.
- Due puntatori a elementi dello stesso array possono essere sottratti l'uno dall'altro per determinare il numero di elementi tra loro.
- È possibile assegnare un puntatore a un altro puntatore se entrambi hanno lo stesso tipo. Fa eccezione il puntatore di tipo `void *` che può rappresentare qualsiasi tipo di puntatore. È possibile assegnare un puntatore `void *` a tutti i tipi di puntatori e un puntatore di qualsiasi tipo a un puntatore `void *`.
- Un puntatore `void *` non può essere dereferenziato.
- È possibile **confrontare i puntatori** usando gli operatori di uguaglianza e relazionali, ma tali confronti non hanno senso, a meno che i puntatori non puntino a elementi dello stesso array. I confronti fra puntatori confrontano gli indirizzi memorizzati nei puntatori.
- Un uso comune del confronto fra puntatori è quello di **determinare se un puntatore è NULL**.

Paragrafo 7.9 Relazioni tra puntatori e array

- Array e puntatori sono intimamente correlati in C e spesso possono essere usati in maniera interscambiabile.
- Al nome di un array si può pensare come a un **puntatore costante**.
- I puntatori possono essere usati per compiere qualsiasi operazione che implichi l'indicizzazione di un array.
- Quando un puntatore punta all'inizio di un array, aggiungere un **offset** al puntatore indica a quale elemento dell'array si deve fare riferimento. Il valore dell'offset è identico all'indice dell'array. Questa è detta notazione puntatore/offset.
- Il nome di un array può essere trattato come un **puntatore** e usato nelle espressioni dell'aritmetica dei puntatori che non tentano di modificare l'indirizzo del puntatore.
- È possibile **indicizzare i puntatori** come gli array. Questa è chiamata notazione puntatore/indice.
- Un parametro di tipo `const char *` tipicamente rappresenta una stringa costante.

Paragrafo 7.10 Array di puntatori

- Gli array possono contenere puntatori. Un uso comune di un array di puntatori è quello di formare un **array di stringhe**. Ogni elemento nell'array è una stringa, ma in C una stringa è essenzialmente un puntatore al suo primo carattere. Così, ogni elemento in un array di stringhe è in realtà un puntatore al primo carattere di una stringa.

Paragrafo 7.12 Puntatori a funzioni

- Un **puntatore a una funzione** contiene l'indirizzo della funzione in memoria. Il nome di una funzione è in realtà l'indirizzo di partenza in memoria del codice che esegue il compito della funzione.
- I puntatori a funzioni possono essere **passati a funzioni**, **restituiti da funzioni**, **memorizzati in array**, **assegnati ad altri puntatori a funzioni** e **confrontati** tra loro per egualianza o inegualianza.
- Un puntatore a una funzione viene dereferenziato per chiamare la funzione. Il puntatore a una funzione può essere usato direttamente come nome della funzione quando si chiama la funzione.
- Un uso comune dei puntatori a funzioni si ha nei sistemi guidati da menu.

Esercizi di autovalutazione

7.1 Completate ciascuna delle seguenti frasi:

- Una variabile puntatore contiene come suo valore l' _____ di un'altra variabile.
- I tre valori che si possono usare per inizializzare un puntatore sono _____, _____ e _____.
- L'unico intero che si può assegnare a un puntatore è _____.

- 7.2 Stabilite se le seguenti affermazioni sono *vere* o *false*. Se la risposta è *false*, spiegate il perché.
- Un puntatore dichiarato `void` può essere dereferenziato.
 - I puntatori a tipi differenti non possono essere assegnati l'uno all'altro senza un'operazione di cast.
- 7.3 Eseguite ognuna delle seguenti operazioni e rispondete alle domande. Supponete che i numeri in virgola mobile con precisione singola siano memorizzati in 4 byte e che l'indirizzo di partenza dell'array sia alla locazione di memoria 1002500. Ogni parte dell'esercizio deve usare i risultati delle parti precedenti dove è opportuno.
- Definite un array di tipo `float` chiamato `numbers` con 10 elementi e inizializzate gli elementi ai valori 0.0, 1.1, 2.2, ..., 9.9. Supponete che la costante simbolica `SIZE` sia stata definita come 10.
 - Definite un puntatore, `nPtr`, che punti a un `float`.
 - Usate un'istruzione `for` e una notazione con indice per gli array per stampare gli elementi dell'array `numbers`. Usate una cifra di precisione a destra del punto decimale.
 - Scrivete due diverse istruzioni che assegnino l'indirizzo di partenza dell'array `numbers` alla variabile puntatore `nPtr`.
 - Stampate gli elementi dell'array `numbers` usando la notazione puntatore/offset con il puntatore `nPtr`.
 - Stampate gli elementi dell'array `numbers` usando la notazione puntatore/offset con il nome dell'array come puntatore.
 - Stampate gli elementi dell'array `numbers` indicizzando il puntatore `nPtr`.
 - Fate riferimento all'elemento 4 dell'array `numbers` usando la notazione con indice per gli array, la notazione puntatore/offset con il nome dell'array come puntatore, la notazione con indice per i puntatori con `nPtr` e la notazione puntatore/offset con `nPtr`.
 - Supponendo che `nPtr` punti all'inizio dell'array `numbers`, a quale indirizzo si fa riferimento con `nPtr + 8`? Quale valore è memorizzato in quella locazione?
 - Supponendo che `nPtr` punti a `numbers[5]`, a quale indirizzo si fa riferimento con `nPtr - 4`? Qual è il valore memorizzato in quella locazione?
- 7.4 Per ognuna delle seguenti operazioni, scrivete un'istruzione che la esegua. Supponete che le variabili in virgola mobile (`float`) `number1` e `number2` siano state definite e che `number1` sia inizializzato a 7.3.
- Definire la variabile `fPtr` come puntatore a un oggetto di tipo `float`.
 - Assegnare l'indirizzo della variabile `number1` alla variabile puntatore `fPtr`.
 - Stampare il valore dell'oggetto puntato da `fPtr`.
 - Assegnare il valore dell'oggetto puntato da `fPtr` alla variabile `number2`.
 - Stampare il valore di `number2`.
 - Stampare l'indirizzo di `number1`. Usare la specifica di conversione `%p`.
 - Stampare l'indirizzo memorizzato in `fPtr`. Usare la specifica di conversione `%p`. Il valore stampato è lo stesso dell'indirizzo di `number1`?
- 7.5 Eseguite ognuna delle seguenti operazioni.
- Scrivete l'intestazione per una funzione chiamata `exchange` che riceve come parametri due puntatori ai numeri in virgola mobile `x` e `y` e non restituisce alcun valore.
 - Scrivete il prototipo di funzione per la funzione di cui alla voce a).
 - Scrivete l'intestazione per una funzione chiamata `evaluate` che restituisce un intero e che riceve come parametri l'intero `x` e un puntatore alla funzione `poly`, che rappresenta una funzione che riceve un parametro intero e restituisce un intero.
 - Scrivete il prototipo di funzione per la funzione di cui alla voce c).
- 7.6 Trovate l'errore in ognuno dei seguenti segmenti di programma. Presupponete quanto segue:

```
int *zPtr; // zPtr farà riferimento all'array z
int *aPtr = NULL;
void *sPtr = NULL;
int number;
int z[5] = {1, 2, 3, 4, 5};
sPtr = z;
```

- a) `++zptr;`
- b) // usa il puntatore per ottenere l'array
`number = zPtr;`
- c) // assegna a number l'elemento 2; si suppone che zPtr sia inizializzato
`number = *zPtr[2];`
- d) // stampa l'intero array z; si suppone che zPtr sia inizializzato
`for (size_t i = 0; i <= 5; ++i) {`
 `printf("%d ", zPtr[i]);`
`}`
- e) // assegna a number il valore a cui punta sPtr
`number = *sPtr;`
- f) `++z;`

Risposte agli esercizi di autovalutazione

- 7.1 a) indirizzo. b) 0, NULL, un indirizzo. c) 0.
- 7.2 a) *Falso*. Un puntatore a void non può essere dereferenziato, perché non c'è modo di sapere esattamente quanti byte di memoria dereferenziare. b) *Falso*. È possibile assegnare puntatori di altri tipi ai puntatori di tipo void e puntatori di tipo void a puntatori di altri tipi.
- 7.3 a) `float numbers[SIZE] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};`
- b) `float *nPtr;`
- c) `for (size_t i = 0; i < SIZE; ++i) {`
 `printf("%.1f ", numbers[i]);`
`}`
- d) `nPtr = numbers;`
`nPtr = &numbers[0];`
- e) `for (size_t i = 0; i < SIZE; ++i) {`
 `printf("%.1f ", *(nPtr + i));`
`}`
- f) `for (size_t i = 0; i < SIZE; ++i) {`
 `printf("%.1f ", *(numbers + i));`
`}`
- g) `for (size_t i = 0; i < SIZE; ++i) {`
 `printf("%.1f ", nPtr[i]);`
`}`
- h) `numbers[4]`
`*(numbers + 4)`
`nPtr[4]`
`*(nPtr + 4)`
- i) L'indirizzo è $1002500 + 8 * 4 = 1002532$. Il valore è 8.8.
- j) L'indirizzo di `numbers[5]` è $1002500 + 5 * 4 = 1002520$.
L'indirizzo di `nPtr - 4` è $1002520 - 4 * 4 = 1002504$.
Il valore in quella locazione è 1.1.
- 7.4 a) `float *fPtr;`
- b) `fPtr = &number1;`
- c) `printf("The value of *fPtr is %f\n", *fPtr);`
- d) `number2 = *fPtr;`
- e) `printf("The value of number2 is %f\n", number2);`
- f) `printf("The address of number1 is %p\n", &number1);`
- g) `printf("The address stored in fptr is %p\n", fPtr);`
- Si, il valore è lo stesso.

- 7.5 a) `void exchange(float *x, float *y)`
 b) `void exchange(float *x, float *y);`
 c) `int evaluate(int x, int (*poly)(int))`
 d) `int evaluate(int x, int (*poly)(int));`
- 7.6 a) Errore: `zPtr` non è stato inizializzato.
 Correzione: inizializzate `zPtr` con `zPtr = z`; prima di usare l'aritmetica dei puntatori.
 b) Errore: il puntatore non è dereferenziato.
 Correzione: cambiate l'istruzione in `number = *zPtr;`
 c) Errore: `zPtr[2]` non è un puntatore e non va dereferenziato.
 Correzione: cambiate `*zPtr[2]` to `zPtr[2]`.
 d) Errore: fare riferimento a un elemento al di fuori dei confini dell'array con l'indicizzazione del puntatore.
 Correzione: cambiate l'operatore `<=` nella condizione del `for` in `<`.
 e) Errore: dereferenziare un puntatore `void`.
 Correzione: per dereferenziare il puntatore si deve prima effettuare un cast a un puntatore intero.
 Cambiare l'istruzione in `number = *((int *) sPtr);`
 f) Errore: cercare di modificare il nome di un array con l'aritmetica dei puntatori.
 Correzione: usate una variabile puntatore invece del nome dell'array per applicare l'aritmetica dei puntatori, oppure indicizzate il nome dell'array per fare riferimento a uno specifico elemento.

Esercizi

- 7.7 Completate ciascuna delle seguenti frasi.
- L'operatore _____ restituisce la locazione in memoria dove è memorizzato il suo operando.
 - L'operatore _____ restituisce il valore dell'oggetto al quale punta il suo operando.
 - Per compiere il passaggio per riferimento quando una variabile diversa da un array viene passata a una funzione, è necessario passare alla funzione l'_____ della variabile.
- 7.8 Stabilite se le seguenti affermazioni sono *vere* o *false*. Se *false*, spiegate perché.
- Non ha senso confrontare due puntatori che puntano ad array differenti.
 - Dal momento che il nome di un array è un puntatore al primo elemento dell'array, i nomi degli array possono essere manipolati precisamente alla stessa maniera dei puntatori.
- 7.9 Eseguite ognuna delle seguenti operazioni e rispondete alle domande. Supponete che gli interi senza segno siano memorizzati in 4 byte e che l'indirizzo di partenza dell'array sia alla locazione in memoria 1002500.
- Definite un array di tipo `int` di 5 elementi chiamato `values` e inizializzate gli elementi con gli interi pari da 2 a 10. Supponete che la costante simbolica `SIZE` sia stata definita come 5.
 - Definite un puntatore `vPtr` che punta a un oggetto di tipo `int`.
 - Stampate gli elementi dell'array `values` usando la notazione con indice degli array. Usate un'istruzione `for` e supponete che la variabile di controllo intera `i` sia stata definita.
 - Scrivete due istruzioni separate che assegnino l'indirizzo di partenza dell'array `values` alla variabile puntatore `vPtr`.
 - Stampate gli elementi dell'array `values` usando la notazione puntatore/offset.
 - Stampate gli elementi dell'array `values` usando la notazione puntatore/offset con il nome dell'array come puntatore.
 - Stampate gli elementi dell'array `values` usando il puntatore all'array con indice.
 - Fate riferimento all'elemento 4 dell'array `values` usando la notazione con indice degli array, la notazione puntatore/offset con il nome dell'array come puntatore, la notazione con indice che usa il puntatore e la notazione puntatore/offset.
 - A quale indirizzo si fa riferimento con `vPtr + 3`? Quale valore è memorizzato in quella locazione?
 - Supponendo che `vPtr` punti a `values[4]`, a quale indirizzo si fa riferimento con `vPtr -= 4`? Quale valore è memorizzato in quella locazione?

7.10 Per ognuna delle seguenti operazioni, scrivete un'istruzione singola che la esegua. Supponete che le variabili intere di tipo long `value1` e `value2` siano state definite e che `value1` sia stata inizializzata a 200000.

- Definire la variabile `lPtr` come puntatore a un oggetto di tipo `long`.
- Assegnare l'indirizzo della variabile `value1` alla variabile puntatore `lPtr`.
- Stampare il valore dell'oggetto puntato da `lPtr`.
- Assegnare il valore dell'oggetto puntato da `lPtr` alla variabile `value2`.
- Stampare il valore di `value2`.
- Stampare l'indirizzo di `value1`.
- Stampare l'indirizzo memorizzato in `lPtr`. Il valore stampato è lo stesso dell'indirizzo di `value1`?

7.11 Eseguite ognuna delle seguenti operazioni.

- Scrivete l'intestazione per la funzione `zero`, la quale riceve il parametro array intero di tipo `long bigIntegers` e non restituisce alcun valore.
- Scrivete il prototipo di funzione per la funzione di cui alla voce a).
- Scrivete l'intestazione per la funzione `add1AndSum`, che riceve il parametro array intero `oneTooSmall` e restituisce un intero.
- Scrivete il prototipo di funzione per la funzione di cui alla voce c).

Nota: gli Esercizi 7.12-7.15 sono ragionevolmente impegnativi. Una volta risolti questi problemi, dovreste essere in grado di implementare facilmente i più popolari giochi di carte.

7.12 (*Mescolare e distribuire le carte: distribuire mani di poker*) Modificate il programma della Figura 7.16 in modo che la funzione che distribuisce le carte distribuisca una mano di poker di cinque carte. Poi scrivete ulteriori funzioni per eseguire le seguenti operazioni.

- Determinare se tra le cinque carte c'è una coppia.
- Determinare se tra le cinque carte vi sono due coppie.
- Determinare se tra le cinque carte ve ne sono tre dello stesso tipo (es. tre fanti).
- Determinare se tra le cinque carte ve ne sono quattro dello stesso tipo (es. quattro assi).
- Determinare se le cinque carte formano un colore (cioè se tutte e cinque le carte sono dello stesso seme).
- Determinare se le cinque carte formano una scala (cioè se le cinque carte hanno valori in sequenza).

7.13 (*Progetto: mescolare e distribuire le carte – Qual è la mano di poker migliore?*) Usate le funzioni sviluppate nell'Esercizio 7.12 per scrivere un programma che distribuisce due mani di poker di cinque carte, le valuta e determina qual è la mano migliore.

7.14 (*Progetto: mescolare e distribuire le carte – Simulare la persona che distribuisce le carte*) Modificate il programma sviluppato nell'Esercizio 7.13 in modo che possa simulare la persona che distribuisce. Le cinque carte sono distribuite "a faccia in giù" cosicché l'altro giocatore non possa vederle. Il programma deve quindi valutare la mano del giocatore simulato che distribuisce e, basandosi sulla bontà della mano, questo giocatore deve prendere una, due o tre carte per sostituire un numero corrispondente di carte non buone della mano originaria. Il programma deve poi valutare di nuovo le carte di chi distribuisce. [Avvertimento: si tratta di un problema difficile!]

7.15 (*Progetto: mescolare e distribuire le carte – Consentire ai giocatori di pescare carte*) Modificate il programma sviluppato nell'Esercizio 7.14 in modo che possa gestire automaticamente le carte del distributore, mentre al giocatore è consentito decidere quali carte sostituire. Il programma deve quindi valutare entrambe le mani e determinare chi vince. Ora usate questo nuovo programma per giocare 20 partite contro il computer. Chi vince più partite, voi o il computer? Fate giocare 20 partite contro il computer a un vostro amico. Chi vince più partite? Basandovi sui risultati di queste partite, fate le opportune modifiche per affinare il vostro programma di gioco del poker (anche questo è un problema difficile). Giocate altre 20 partite. Il vostro programma modificato gioca una partita migliore?

7.16 (Modifica al programma per mescolare e distribuire le carte: mescolamento a elevate prestazioni) Nel programma della Figura 7.16 abbiamo intenzionalmente usato un algoritmo inefficiente per mescolare le carte che introduceva la possibilità di posposizione indefinita. In questo esercizio svilupperete un algoritmo a elevate prestazioni per mescolare le carte che evita la posposizione indefinita.

Modificate il programma della Figura 7.16 come segue. Cominciate inizializzando l'array `deck` come mostrato di seguito:

Array non mescolato													
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	14	15	16	17	18	19	20	21	22	23	24	25	26
2	27	28	29	30	31	32	33	34	35	36	37	38	39
3	40	41	42	43	44	45	46	47	48	49	50	51	52

Modificate la funzione `shuffle` per effettuare un'iterazione riga per riga e colonna per colonna lungo l'array, visitando i vari elementi una volta soltanto. Ogni elemento deve essere scambiato con un elemento dell'array selezionato a caso. Stampate l'array risultante per verificare se il mazzo di carte è mescolato in modo soddisfacente. Di seguito abbiamo un esempio di valori mescolati:

Esempio di array mescolato													
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	19	40	27	25	36	46	10	34	35	41	18	2	44
1	13	28	14	16	21	38	8	11	31	17	24	7	1
2	12	33	15	42	43	23	45	3	29	32	4	47	26
3	50	38	52	39	48	51	9	5	37	49	22	6	28

Per assicurarvi che sia mescolato in modo soddisfacente, fate sì che il vostro programma chiami diverse volte la funzione `shuffle`.

Sebbene l'approccio descritto in questo problema migliori l'algoritmo per mescolare le carte, l'algoritmo di distribuzione richiede ancora la ricerca nell'array `deck` per la carta 1, poi per la carta 2, poi per la carta 3 e così via. Peggio ancora, anche dopo che l'algoritmo localizza e dà una carta, questo continua a cercarla nel resto del mazzo. Modificate il programma della Figura 7.16 così che, una volta che una carta è stata localizzata e data, non vengano fatti ulteriori tentativi per cercarla e il programma proceda immediatamente a distribuire la carta successiva. Nel Capitolo 10 svilupperemo un algoritmo per mescolare le carte che richiede soltanto un'operazione per carta.

7.17 Cosa fa questo programma, supponendo che l'utente inserisca due stringhe di uguale lunghezza?

```

1 // ex07_19.c
2 // Cosa fa questo programma?
3 #include <stdio.h>
4 #define SIZE 80
5
6 void mystery1(char *s1, const char *s2); // prototipo
7
8 int main(void) {

```

```

9  char string1[SIZE]; // crea un array di char
10 char string2[SIZE]; // crea un array di char
11
12 puts("Enter two strings: ");
13 scanf("%39s%39s", string1, string2);
14 mystery1(string1, string2);
15 printf("%s", string1);
16 }
17
18 // Cosa fa questa funzione?
19 void mystery1(char *s1, const char *s2) {
20     while (*s1 != '\0') {
21         ++s1;
22     }
23
24     for (; *s1 = *s2; ++s1, ++s2) {
25         ; // istruzione vuota
26     }
27 }
```

7.18 Cosa fa questo programma?

```

1 // ex07_20.c
2 // Cosa fa questo programma?
3 #include <stdio.h>
4 #define SIZE 80
5
6 size_t mystery2(const char *s); // prototipo
7
8 int main(void) {
9     char string[SIZE]; // crea un array di char
10
11    puts("Enter a string: ");
12    scanf("%79s", string);
13    printf("%d\n", mystery2(string));
14 }
15
16 // Cosa fa questa funzione?
17 size_t mystery2(const char *s) {
18     size_t x;
19
20     // effettua un ciclo lungo la stringa
21     for (x = 0; *s != '\0'; ++s) {
22         ++x;
23     }
24
25     return x;
26 }
```

7.19 Trovate l'errore in ognuno dei seguenti segmenti di programma. Se l'errore può essere corretto, spiegatelo.

a) `int *number;
printf("%d\n", *number);`

```

b) float *realPtr;
   long *integerPtr;
   integerPtr = realPtr;
c) int * x, y;
   x = y;
d) char s[] = "this is a character array";
   int count;
   for (; *s != ; ++s) {
      printf("%c ", *s);
   }
e) short *numPtr, result;
   void *genericPtr = numPtr;
   result = *genericPtr + 7;
f) float x = 19.34;
   float xPtr = &x;
   printf("%f\n", xPtr);
g) char *s;
   printf("%s\n", s);

```

7.20 (Attraversamento del labirinto) La griglia seguente è la rappresentazione con un array bidimensionale di un labirinto. I simboli # rappresentano le pareti del labirinto e i punti (.) rappresentano dei quadrati nei possibili percorsi attraverso il labirinto.

```

# # # # # # # # # #
# . . . # . . . . . #
. . # . # # # # . #
# # # . # . . . . # .
# . . . . # # # . # .
# # # # . # . # . # .
# . . # . # . # . # .
# # . # . # . # . # .
# . . . . . . . . # .
# # # # # . # # # . #
# . . . . . # . . . #
# # # # # # # # # #

```

Nella pagina di Wikipedia https://en.wikipedia.org/wiki/Maze_solving_algorithm sono elencati numerosi algoritmi per trovare l'uscita da un labirinto. Un semplice algoritmo per attraversare un labirinto garantisce di trovare l'uscita (supponendo che ve ne sia una). Mettete la mano destra sulla parete alla vostra destra e iniziate a camminare in avanti. Non togliete mai la mano dal muro. Se il labirinto gira a destra, seguite la parete alla destra. Se non toglierete la mano dal muro, giungerete alla fine all'uscita del labirinto. Se non vi è un'uscita, alla fine ritornerete al punto di partenza. Può darsi che vi sia un percorso più breve di quello che avete trovato, ma avete la garanzia di uscire dal labirinto.

Scrivete una funzione ricorsiva `mazeTraverse` per attraversare il labirinto. La funzione deve ricevere come argomenti un array di caratteri 12-per-12 che rappresenta il labirinto e il punto di partenza del labirinto. Mentre `mazeTraverse` tenta di trovare l'uscita dal labirinto, deve mettere il carattere X in ogni quadrato nel percorso. La funzione deve stampare il labirinto dopo ogni movimento, così l'utente può osservare come viene risolto il problema del suo attraversamento.

7.21 (*Generare labirinti in modo casuale*) Scrivete una funzione `mazeGenerator` che riceve come argomento un array di caratteri 12-per-12 bidimensionale e produce a caso un labirinto. La funzione deve anche fornire i punti di partenza e di arrivo del labirinto. Provate la vostra funzione `mazeTraverse` dell'Esercizio 7.20 usando diversi labirinti generati a caso.

7.22 (*Labirinti di ogni dimensione*) Generalizzate le funzioni `mazeTraverse` e `mazeGenerator` degli Esercizi 7.20 e 7.21 per trattare labirinti di qualsiasi larghezza e altezza.

7.23 Cosa fa questo programma, supponendo che l'utente inserisca due stringhe della stessa lunghezza?

```

1 // ex07_26.c
2 // Cosa fa questo programma?
3 #include <stdio.h>
4 #define SIZE 80
5
6 int mystery3(const char *s1, const char *s2); // prototipo
7
8 int main(void) {
9     char string1[SIZE]; // crea un array di char
10    char string2[SIZE]; // crea un array di char
11
12    puts("Enter two strings: ");
13    scanf("%79s%79s", string1, string2);
14    printf("The result is %d\n", mystery3(string1, string2));
15 }
16
17 int mystery3(const char *s1, const char *s2) {
18     int result = 1;
19
20     for (; *s1 != '\0' && *s2 != '\0'; ++s1, ++s2) {
21         if (*s1 != *s2) {
22             result = 0;
23         }
24     }
25
26     return result;
27 }
```

Array di puntatori a funzioni

7.24 (*Array di puntatori a funzioni*) Riscrivete il programma della Figura 6.17 per usare un'interfaccia guidata da menu. Il programma deve offrire all'utente quattro opzioni, come segue:

<pre> Enter a choice: 0 Print the array of grades 1 Find the minimum grade 2 Find the maximum grade 3 Print the average on all tests for each student 4 End program </pre>
--

Una restrizione all'uso di array di puntatori a funzioni è costituita dal fatto che tutti i puntatori devono avere lo stesso tipo. I puntatori devono essere a funzioni dello stesso tipo di ritorno che ricevono argomenti dello stesso tipo. Per questa ragione, le funzioni nella Figura 6.17 devono essere modificate, in modo che ognuna restituiscia lo stesso tipo di dato e riceva gli stessi parametri. Modificate le funzioni `minimum` e `maximum` in modo che stampino il valore minimo o massimo e non restituiscano niente. Per l'opzione 3, modificate la funzione `average` della Figura 6.17 per stampare la media per ogni studente (non per uno studente specifico). La funzione `average` non deve restituire niente e deve ricevere gli stessi parametri di `printArray`, `minimum` e `maximum`. Memorizzate i puntatori alle quattro funzioni nell'array `processGrades` e usate la scelta fatta dall'utente come indice dell'array per chiamare ogni funzione.

7.25 (Calcolo della circonferenza del cerchio, dell'area del cerchio o del volume della sfera usando puntatori a funzioni) Usando le tecniche che avete appreso nella Figura 7.18, realizzate un programma guidato da menu che permetta all'utente di scegliere se calcolare la circonferenza di un cerchio, l'area di un cerchio o il volume di una sfera. Il programma deve poi leggere il valore del raggio fornito dall'utente, eseguire il calcolo appropriato e stampare il risultato. Usate un array di puntatori a funzioni in cui ogni puntatore rappresenta una funzione che restituisce `void` e riceve un parametro `double`. Le funzioni corrispondenti devono stampare ognuna un messaggio che indica quale calcolo è stato eseguito, il valore del raggio e il risultato del calcolo.

7.26 (Calcolatore che usa puntatori a funzioni) Usando le tecniche che avete imparato nella Figura 7.18, realizzate un programma guidato da menu che permetta all'utente di scegliere se sommare, sottrarre, moltiplicare o dividere due numeri. Il programma deve quindi leggere due valori `double` forniti dall'utente, effettuare il calcolo appropriato e stampare il risultato. Usate un array di puntatori a funzioni in cui ogni puntatore rappresenta una funzione che restituisce `void` e riceve due parametri `double`. Le funzioni corrispondenti devono ognuna stampare un messaggio che indica quale calcolo è stato eseguito, i valori dei parametri e il risultato del calcolo.

7.27 (Calcolatore delle emissioni di anidride carbonica) Usando array di puntatori a funzioni, come avete imparato in questo capitolo, potete specificare un insieme di funzioni che vengono chiamate con gli stessi tipi di argomenti e restituiscono lo stesso tipo di dati. Governi e aziende di tutto il mondo si stanno sempre più interessando alle emissioni di CO₂ (rilasci annuali di biossido di carbonio nell'atmosfera) da parte di edifici che bruciano vari tipi di combustibili per il riscaldamento, di veicoli che bruciano carburanti per i loro motori e così via. Molti scienziati ritengono questi gas con effetto serra responsabili del fenomeno chiamato riscaldamento globale. Create tre funzioni che permettano di calcolare l'emissione di CO₂, rispettivamente, di un edificio, di un'automobile e di una bicicletta. Ogni funzione deve ricevere in ingresso dall'utente i dati corretti, poi deve calcolare e stampare l'emissione di CO₂. (Consultate alcuni siti web che spiegano come calcolare le emissioni di CO₂.) Ogni funzione non deve ricevere alcun parametro e deve restituire `void`. Scrivete un programma che richieda all'utente di inserire il tipo di emissione di CO₂ da calcolare, quindi chiami la funzione corrispondente tramite l'array di puntatori a funzioni. Per ogni tipo di emissione di CO₂, stampate alcune informazioni e la quantità di emissione di CO₂ dell'oggetto.

Paragrafo speciale: costruite il vostro computer come macchina virtuale

Nei prossimi esercizi ci discosteremo temporaneamente dal mondo della programmazione con linguaggi di alto livello. "Apriremo" un computer e guarderemo la sua struttura interna. Introdurremo la programmazione in linguaggio macchina, con il quale scriveremo diversi programmi. Affinché questa risulti un'esperienza particolarmente preziosa, costruiremo quindi una *simulazione* di questo computer basata sul software sulla quale potrete eseguire i vostri programmi in linguaggio macchina! Questo tipo di simulazione di un computer viene spesso chiamata **macchina virtuale**.

7.28 (Programmazione in linguaggio macchina) Creiamo un computer che chiameremo Simpletron. Come indica il suo nome, si tratta di una macchina semplice ma, come presto vedremo, allo stesso tempo potente. Il Simpletron esegue programmi scritti nell'unico linguaggio che comprende direttamente, vale a dire il **Simpletron Machine Language** o, abbreviato, **SML**.

Il Simpletron contiene un **accumulatore**, un “registro speciale” in cui le informazioni sono depositate prima che il computer le usi nei calcoli o le esamini in vari modi. Tutte le informazioni nel Simpletron sono trattate in termini di **parole**. Una parola è un numero decimale di quattro cifre con segno come +3364, -1293, +0007, -0001 e così via. Il Simpletron è dotato di una memoria di 100 parole e a queste parole si fa riferimento con i loro numeri di locazione 00, 01, ..., 99.

Prima di eseguire un programma in SML, dobbiamo **caricarlo** o metterlo in memoria. La prima istruzione di ogni programma in SML è sempre posta nella locazione 00.

Ogni **istruzione scritta in SML** occupa una parola di memoria del Simpletron, così le istruzioni sono numeri decimali di quattro cifre con un segno. Supponiamo che il segno di un’istruzione in SML sia sempre il più, ma il segno di una parola di dati può essere il più oppure il meno. Ogni locazione nella memoria del Simpletron può contenere un’istruzione, un dato usato da un programma o un’area inutilizzata (e quindi indefinita) della memoria. Le prime due cifre di ogni istruzione in SML sono il **codice operativo**, il quale specifica l’operazione da eseguire. I codici operativi del SML sono riepilogati nella seguente tabella.

Codice operativo	Significato
<i>Operazioni di input/output:</i>	
#define READ 10	Leggi una parola immessa da tastiera e inseriscila in una specifica locazione in memoria.
#define WRITE 11	Scrivi su schermo una parola contenuta in una specifica locazione in memoria.
<i>Operazioni di caricamento/memorizzazione:</i>	
#define LOAD 20	Carica nell’accumulatore la parola contenuta in una specifica locazione in memoria.
#define STORE 21	Archivia una parola contenuta nell’accumulatore in una specifica locazione di memoria.
<i>Operazioni aritmetiche:</i>	
#define ADD 30	Aggiungi una parola contenuta in una specifica locazione di memoria alla parola nell’accumulatore (lasciando il risultato nell’accumulatore).
#define SUBTRACT 31	Sottrai una parola contenuta in una specifica locazione di memoria dalla parola nell’accumulatore (lasciando il risultato nell’accumulatore).
#define DIVIDE 32	Dividi una parola contenuta in una specifica locazione di memoria per la parola nell’accumulatore (lasciando il risultato nell’accumulatore).
#define MULTIPLY 33	Moltiplica la parola contenuta in una specifica locazione di memoria per la parola nell’accumulatore (lasciando il risultato nell’accumulatore).
<i>Operazioni di trasferimento del controllo:</i>	
#define BRANCH 40	Salta a una specifica locazione di memoria.
#define BRANCHNEG 41	Salta a una specifica locazione di memoria se l’accumulatore è negativo.
#define BRANCHZERO 42	Salta a una specifica locazione di memoria se l’accumulatore è zero.
#define HALT 43	Halt, cioè il programma ha completato il suo compito.

Le ultime due cifre di un’istruzione in SML costituiscono l’**operando**, ovvero la locazione di memoria contenente la parola alla quale si applica l’operazione.

Esempio di programma in SML che somma due numeri

Ora consideriamo diversi semplici programmi in SML. Il seguente programma in SML legge due numeri dalla tastiera e calcola e stampa la loro somma.

Locazione	Numero	Istruzione
00	+1007	(Read A)
01	+1008	(Read B)
02	+2007	(Load A)
03	+3008	(Add B)
04	+2109	(Store C)
05	+1109	(Write C)
06	+4300	(Halt)
07	+0000	(Variable A)
08	+0000	(Variable B)
09	+0000	(Result C)

L'istruzione +1007 legge il primo numero dalla tastiera e lo inserisce nella locazione 07. Poi +1008 legge il numero successivo e lo colloca nella locazione 08. L'istruzione *load*, +2007, copia il primo numero nell'accumulatore. L'istruzione *add*, +3008, addiziona il secondo numero al numero nell'accumulatore. *Tutte le istruzioni aritmetiche del SML lasciano i loro risultati nell'accumulatore*. L'istruzione *store*, +2109, copia il risultato contenuto nell'accumulatore nella locazione di memoria 09, dalla quale l'istruzione *write*, +1109, poi riceve il numero e lo stampa come numero decimale di quattro cifre con segno. L'istruzione *halt*, +4300, termina l'esecuzione.

Esempio di programma in SML che determina il maggiore tra due valori

Il seguente programma in SML legge due numeri dalla tastiera e determina e stampa il valore maggiore.

Locazione	Numero	Istruzione
00	+1009	(Read A)
01	+1010	(Read B)
02	+2009	(Load A)
03	+3110	(Subtract B)
04	+4107	(Branch negative to 07)
05	+1109	(Write A)
06	+4300	(Halt)
07	+1110	(Write B)
08	+4300	(Halt)
09	+0000	(Variable A)
10	+0000	(Variable B)

L'istruzione +4107 è un trasferimento condizionato del controllo, come un'istruzione *if*.

Adesso scrivete alcuni programmi in SML per eseguire ognuna delle seguenti operazioni.

- Usare un ciclo controllato da sentinella per leggere interi positivi e per calcolare e stampare la loro somma.
- Usare un ciclo controllato da contatore per leggere sette numeri, alcuni positivi e alcuni negativi. Calcolare e stampare la loro media.
- Leggere una serie di numeri e determinare e stampare il maggiore. Il primo numero letto indica quanti numeri vanno elaborati.

7.29 (Un simulatore di computer) Sulle prime può sembrare esagerato, ma in questo problema costruirete il vostro computer. No, non unirete insieme dei componenti. Piuttosto, userete la potente tecnica della **simulazione software** per creare un **modello software** del Simpletron. Non sarete delusi. Il vostro simulatore del Simpletron trasformerà il computer che state usando in un Simpletron, e sarete realmente in grado di eseguire, testare e correggere i programmi in SML che avete scritto nell'Esercizio 7.28!

Quando eseguite il vostro simulatore del Simpletron, questo deve iniziare l'elaborazione stampando:

```
*** Welcome to Simpletron ***
*** Please enter your program one instruction ***
*** (or data word) at a time. I will type the ***
*** location number and a question mark (?). ***
*** You then type the word for that location. ***
*** Type the sentinel -99999 to stop entering ***
*** your program. ***
```

Simulate la memoria del Simpletron con un array unidimensionale `memory` di 100 elementi. Ora supponiamo che il simulatore sia in esecuzione e analizziamo il codice che si svilupperà con esso mentre inseriamo il programma dell'Esempio 2 dell'Esercizio 7.28:

```
00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300
07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
11 ? -99999
*** Program loading completed ***
*** Program execution begins ***
```

Il programma in SML è stato adesso collocato (o caricato) nell'array `memory`. Ora il Simpletron esegue il programma in SML. Inizia con l'istruzione nella locazione 00 e continua in modo sequenziale, a meno che non venga diretto in qualche altra parte del programma con un trasferimento del controllo.

Usate la variabile `accumulator` per rappresentare il registro dell'accumulatore. Usate la variabile `instructionCounter` per tenere traccia della locazione in memoria (da 00 a 99) che contiene l'istruzione che viene eseguita. Usate la variabile `operationCode` per memorizzare l'operazione correntemente in esecuzione (le due cifre a sinistra della parola dell'istruzione). Usate la variabile `operand` per memorizzare il numero della locazione di memoria su cui opera l'istruzione corrente. Così, se un'istruzione ha un `operand`, questo è formato dalle due cifre più a destra dell'istruzione correntemente in esecuzione. Non fate eseguire le istruzioni

direttamente dalla memoria. Invece, trasferite la successiva istruzione che deve essere eseguita dalla memoria in una variabile chiamata `instructionRegister`. Poi “prelevate” le due cifre a sinistra e mettetele nella variabile `operationCode` e “prelevate” le due cifre a destra e mettetele in `operand`.

Quando il Simpletron inizia l'esecuzione, i suoi registri sono inizializzati come segue:

accumulator	+0000
instructionCounter	00
instructionRegister	+0000
operationCode	00
operand	00

Ora “seguiamo da vicino” l’esecuzione della prima istruzione in SML, +1009 nella locazione di memoria 00. Questo processo di esecuzione è chiamato **ciclo di esecuzione delle istruzioni**.

L'指令计数器 (instruction counter) ci dice la locazione della prossima istruzione da eseguire. Preleviamo i contenuti da quella locazione di memoria tramite l'istruzione `in C`.

```
instructionRegister = memory[instructionCounter];
```

Il codice operativo e l'operando sono estratti dal registro delle istruzioni tramite le istruzioni

```
operationCode = instructionRegister / 100;  
operand = instructionRegister % 100;
```

Adesso il Simpletron deve verificare che il codice operativo sia proprio un *read* (invece che un *write*, un *load* e così via). Un'istruzione *switch* distingue tra le dodici operazioni del SML. L'istruzione *switch* simula il comportamento di varie istruzioni del SML nel modo seguente (lasciamo le altre al lettore):

```

read:    scanf("%d", &memory[operand]);
load:   accumulator = memory[operand];
add:    accumulator += memory[operand];
Varie istruzioni di salto: Le esamineremo a breve.
halt:   Questa istruzione scrive il messaggio

```

*** Simpletron execution terminated ***

poi vengono stampati il nome e i contenuti di ogni registro, come pure i contenuti completi delle 100 locazioni di memoria. Una tale stampa completa viene chiamata **dump** (ovvero "immagine della memoria") del computer. Per aiutarvi a programmare la vostra funzione dump, l'output seguente ne mostra un esempio.

```

REGISTERS:
accumulator           +00000
instructionCounter    00
instructionRegister   +00000
operationCode         00
operand               00

MEMORY:
  0   1   2   3   4   5   6   7   8   9
0  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
10 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
20 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
30 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
40 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
50 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
60 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
70 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
80 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
90 +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000

```

Un dump successivo all'esecuzione di un programma sul Simpletron mostra i valori effettivi delle istruzioni e i valori dei dati al momento in cui l'esecuzione è terminata. Potete stampare gli zeri iniziali davanti a un intero che è più corto della sua ampiezza di campo mettendo il flag di formattazione 0 prima del valore dell'ampiezza del campo nello specificatore di formato, come in "%02d". Potete anche porre un segno + o - prima del valore. Così per stampare un numero della forma +0000 potete usare lo specificatore di formato "%+05d".

Procediamo con l'esecuzione della prima istruzione del nostro programma, ossia +1009 memorizzata nella locazione 00. Come abbiamo indicato, l'istruzione `switch` la simula eseguendo l'istruzione

```
scanf ("%d", &memory[operand]);
```

Prima che `scanf` sia eseguita, va stampato sullo schermo un punto interrogativo (?) per richiedere l'input all'utente. Il Simpletron aspetta che l'utente scriva un valore e poi prema il tasto *Invio*. Il valore viene allora letto e memorizzato nella locazione 09.

A questo punto la simulazione della prima istruzione è completata. Tutto quel che resta è predisporre il Simpletron a eseguire l'istruzione successiva. Poiché l'istruzione appena eseguita non era un trasferimento del controllo, si deve semplicemente incrementare il registro contatore delle istruzioni come segue:

```
++instructionCounter;
```

Questo completa l'esecuzione simulata della prima istruzione. L'intero processo (ossia il **ciclo di esecuzione dell'istruzione**) ricomincia da capo con il **prelevamento (fetch)** della successiva istruzione da eseguire.

Adesso osserviamo come sono simulate le istruzioni di salto, cioè i trasferimenti di controllo. Tutto quello che dobbiamo fare è aggiustare adeguatamente il valore nel contatore delle istruzioni. Pertanto, l'istruzione di salto non condizionato (40) è simulata all'interno del `switch` come

```
instructionCounter = operand;
```

L'istruzione di salto condizionato "salta se l'accumulatore è zero" è simulata come

```
if (accumulator == 0) {
    instructionCounter = operand;
}
```

A questo punto dovreste implementare il vostro simulatore del Simpletron ed eseguire i programmi in SML che avete scritto nell'Esercizio 7.28. Potete perfezionare il linguaggio SML con ulteriori caratteristiche e implementarle nel vostro simulatore. Nell'Esercizio 7.30 sono elencati diversi possibili miglioramenti.

Il vostro simulatore deve controllare vari tipi di errori. Durante la **fase di caricamento del programma**, per esempio, ogni numero che l'utente inserisce nella memoria del Simpletron deve stare nell'intervallo da -9999 a +9999. Il simulatore deve usare un ciclo `while` per verificare che ogni numero inserito sia compreso in questo intervallo e, se non lo è, continuare a chiedere all'utente di reinserire il numero finché non viene inserito un numero corretto.

Durante la fase di esecuzione il vostro simulatore deve controllare gli errori gravi, come i tentativi di **dividere per zero**, i tentativi di eseguire **codici operativi non validi** e **overflow dell'accumulatore** (vale a dire operazioni aritmetiche che producono valori maggiori di +9999 o inferiori a -9999). Tali gravi errori sono chiamati errori irreversibili. Quando viene individuato un errore irreversibile, deve essere stampato un messaggio di errore come:

```
*** Attempt to divide by zero
*** Simpletron execution abnormally terminated ***
```

e un dump completo del computer nel formato che abbiamo esaminato precedentemente. Questo aiuterà l'utente a localizzare l'errore nel programma.

Nota di implementazione: quando implementate il simulatore del Simpletron, definite l'array `memory` e tutti i registri come variabili in `main`. Il programma deve contenere altre tre funzioni: `load`, `execute` e `dump`. La funzione `load` legge le istruzioni in SML dall'utente alla tastiera. (Dopo aver studiato l'elaborazione di file nel Capitolo 11, sarete in grado di leggere le istruzioni in SML da un file.) La funzione `execute` esegue il programma in SML caricato nell'array `memory`. La funzione `dump` stampa i contenuti di `memory` e di tutti i registri memorizzati nelle variabili di `main`. Passate l'array `memory` e i registri alle altre funzioni quando ciò ri-

sulta necessario per l'esecuzione dei loro compiti. Le funzioni `load` ed `execute` hanno bisogno di modificare le variabili definite in `main`, per cui dovete passare quelle variabili a queste funzioni per riferimento usando i puntatori. Dovrete modificare le istruzioni che abbiamo visto nel corso della descrizione di questo problema per usare la notazione appropriata con i puntatori.

7.30 (*Modifiche al simulatore del Simpletron*) In questo esercizio proponiamo diverse modifiche e miglioramenti al simulatore del Simpletron dell'Esercizio 7.29. Negli Esercizi 12.24 e 12.25 proporremo la costruzione di un compilatore che trasforma i programmi scritti in un linguaggio di programmazione di alto livello (una variante del BASIC) nel linguaggio macchina del Simpletron. Potrebbero essere necessari alcuni dei miglioramenti e delle modifiche che seguono per eseguire i programmi prodotti dal compilatore.

- a) Estendete la memoria del simulatore del Simpletron in modo da contenere 1.000 locazioni di memoria (da 000 a 999) per consentire al Simpletron di gestire programmi più grandi.
- b) Fate sì che il simulatore possa calcolare il resto della divisione. Questo richiede un'ulteriore istruzione del linguaggio macchina del Simpletron.
- c) Fate sì che il simulatore possa calcolare l'elevamento a potenza. Questo richiede un'ulteriore istruzione del linguaggio macchina del Simpletron.
- d) Modificate il simulatore in modo da usare valori esadecimali invece che valori interi per rappresentare le istruzioni del linguaggio macchina del Simpletron. Nell'Appendice E (disponibile sulla piattaforma MyLab) parleremo degli esadecimali.
- e) Modificate il simulatore per permettere l'output di un newline. Questo richiede un'ulteriore istruzione del linguaggio macchina del Simpletron.
- f) Modificate il simulatore per elaborare valori in virgola mobile oltre che valori interi.
- g) Modificate il simulatore per rilevare errori logici di divisione per 0.
- h) Modificate il simulatore per rilevare gli errori di overflow aritmetico.
- i) Modificate il simulatore per gestire l'input di stringhe. [Suggerimento: ogni parola del Simpletron può essere divisa in due parti, ognuna contenente un intero di due cifre. Ogni intero di due cifre rappresenta l'equivalente decimale ASCII di un carattere. Aggiungete un'istruzione nel linguaggio macchina che legge una stringa e la memorizza a partire da una specifica locazione di memoria del Simpletron. La prima metà della parola in quella locazione conterrà come valore il conteggio del numero di caratteri nella stringa (cioè la lunghezza della stringa). Ogni metà di parola in successione contiene un carattere ASCII rappresentato con due cifre decimali. La nuova istruzione in linguaggio macchina deve convertire ogni carattere nel suo equivalente valore ASCII e assegnarlo a una metà di parola.]
- j) Modificate il simulatore per gestire l'output di stringhe memorizzate nel formato di cui al punto (g). [Suggerimento: aggiungete un'istruzione nel linguaggio macchina che stampa una stringa iniziando da una specifica locazione di memoria del Simpletron. La prima metà della parola in quella locazione è la lunghezza della stringa in caratteri. Ogni metà di parola in successione contiene un carattere ASCII espresso come due cifre decimali. La nuova istruzione in linguaggio macchina controlla la lunghezza e stampa la stringa traducendo ogni numero di due cifre nel suo carattere equivalente.]

Paragrafo speciale – Programmazione di sistemi embedded

Caso pratico: robotica con il simulatore Webots

7.31 Webots^{5,6} è un simulatore di robotica open source, 3D e a colori, con qualità grafica al livello delle console per videogiochi. Permette di creare una **realità virtuale** nella quale i robot interagiscono con simulazioni di ambienti del mondo reale. È eseguibile su Windows, macOS e Linux. Questo simulatore è ampiamente utilizzato nei campi dell'industria e della ricerca per testare la fattibilità dei robot e per sviluppare il software di controllo dei robot. Webots usa una licenza open source Apache. La citazione seguente è tratta dalla loro pagina web relativa alla licenza:⁷

5. "Webots Open Source Robot Simulator." Accesso 11 dicembre 2020. <https://cyberbotics.com>.

6. Le schermate Webots di questo caso pratico sono soggette al Copyright 2020 Cyberbotics Ltd., concesso in base alla licenza Apache, Versione 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>).

7. "Webots User Guide R2020b revision 2 ~ License Agreement." Accesso 14 dicembre 2020. <https://cyberbotics.com/doc/guide/webots-license-agreement>.

"Webots è rilasciato secondo i termini del contratto di licenza Apache 2.0. Apache 2.0 è [sic] una licenza open source adatta per l'uso a livello industriale, non contaminante, permissiva, che garantisce a chiunque il diritto di utilizzo del codice sorgente di un software, a titolo gratuito, per qualsiasi scopo, comprese le applicazioni commerciali."

Webots è stato sviluppato per la prima volta nel 1996 alla École polytechnique fédérale de Lausanne (EPFL). Nel 1998 fu fondata la società spin-off di EPFL Cyberbotics Ltd. che ha preso in carico lo sviluppo del simulatore Webots. Fino al 2018, Webots veniva venduto come software a licenza proprietaria. Nel 2018, Cyberbotics ha reso Webots open source con licenza Apache 2.0.^{8,9}

La robotica è un argomento che non viene generalmente trattato nei manuali di introduzione alla programmazione, ma con Webots diventa semplice affrontare questo tema. Webots include simulazioni per dozzine di popolari robot del mondo reale che camminano, volano, rotolano, guidano e altro ancora. Per un elenco aggiornato dei robot inclusi, visitate

<https://cyberbotics.com/doc/guide/robots>

Ambiente di sviluppo autonomo

Webots è un ambiente di sviluppo autonomo per la simulazione di robot che ha tutto quello che vi serve per iniziare a sviluppare e sperimentare nella robotica. Include:

- un'area di simulazione interattiva in 3D per visualizzare le simulazioni e interagire con esse;
- un editor di codice per vedere il codice delle simulazioni, modificarlo e scriverne di nuovo;
- compilatori e interpreti che vi consentono di scrivere il codice di Webots nei linguaggi C, C++, Java, Python e MatLab.

Potete modificare con facilità le simulazioni esistenti ed eseguire di nuovo il loro codice per vedere gli effetti delle vostre modifiche. Potete anche sviluppare simulazioni completamente nuove, come farete in questo caso pratico.

Installare Webots

Per prima cosa, controllate i requisiti di sistema di Webots all'indirizzo

<https://cyberbotics.com/doc/guide/system-requirements>

Potete scaricare il file di installazione di Webots per Windows, macOS o Linux dall'home page del sito di Cyberbotics:

<https://cyberbotics.com/>

Una volta installato il file, eseguitelo e seguite le istruzioni sullo schermo.

Tour guidato

Quando avrete terminato l'installazione, eseguite l'applicazione Webots sul vostro sistema. Potete trovare una rapida panoramica di numerose simulazioni di robotica incluse e funzionalità dei robot nel tour guidato di Webots. Per accedervi selezionate Help > Webots Guided Tour... (la prima volta che viene aperto l'ambiente Webots, il tour guidato inizia automaticamente). Compare una finestra (Figura 7.19) nella quale dovete selezionare la casella di controllo Auto e fare clic su Next. Comincerà così il tour guidato automatico, che vi farà vedere brevemente ogni dimostrazione, per poi passare alla successiva. La finestra Guided Tour - Webots visualizza una descrizione sintetica di ciascuna simulazione. La Figura 7.19 mostra la descrizione della terza simulazione, mostrata nella Figura 7.20.

8. "Cyberbotics." Accesso 13 dicembre 2020. <https://cyberbotics.com/#cyberbotics>.

9. "Webots." Accesso 13 dicembre 2020. <https://en.wikipedia.org/wiki/Webots>.

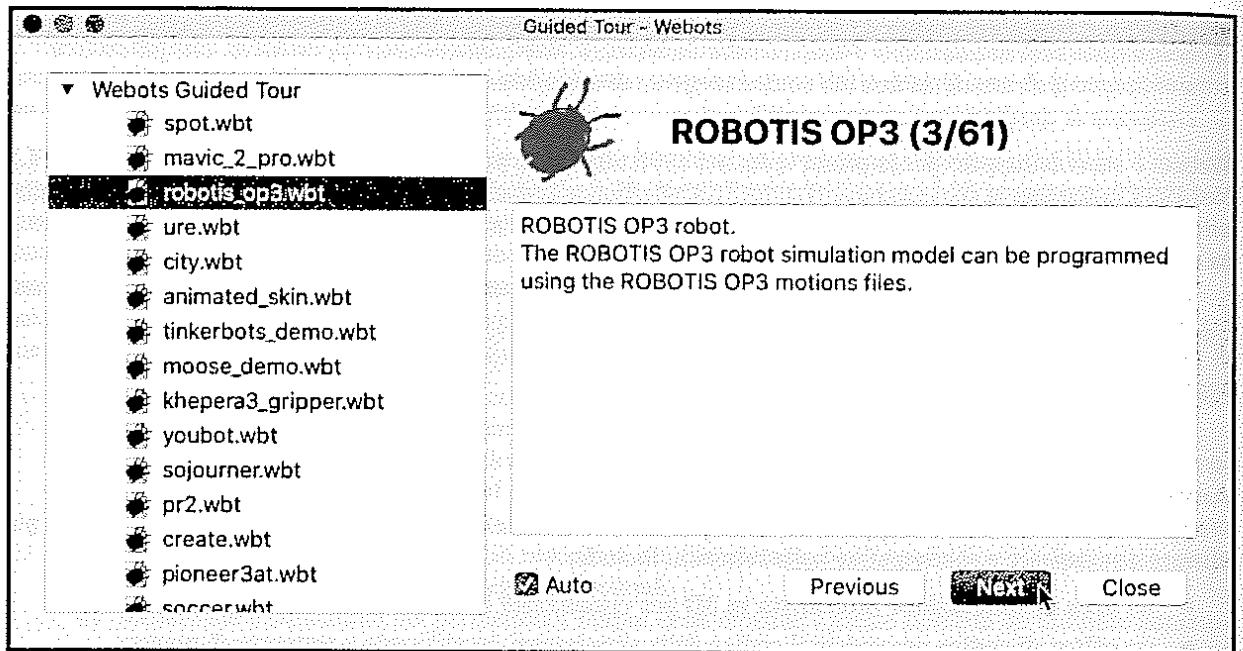


Figura 7.19 Una breve descrizione del robot ROBOTIS OP3 mostrato nella Figura 7.20. [Le schermate di questo caso pratico sono soggette al Copyright 2020 Cyberbotics Ltd., concesso in base alla licenza Apache, Versione 2.0 (la "Licenza"); è vietato l'utilizzo di questo file, se non per quanto previsto dalla Licenza. È possibile consultare una copia della Licenza sul sito <http://www.apache.org/licenses/LICENSE-2.0>.]

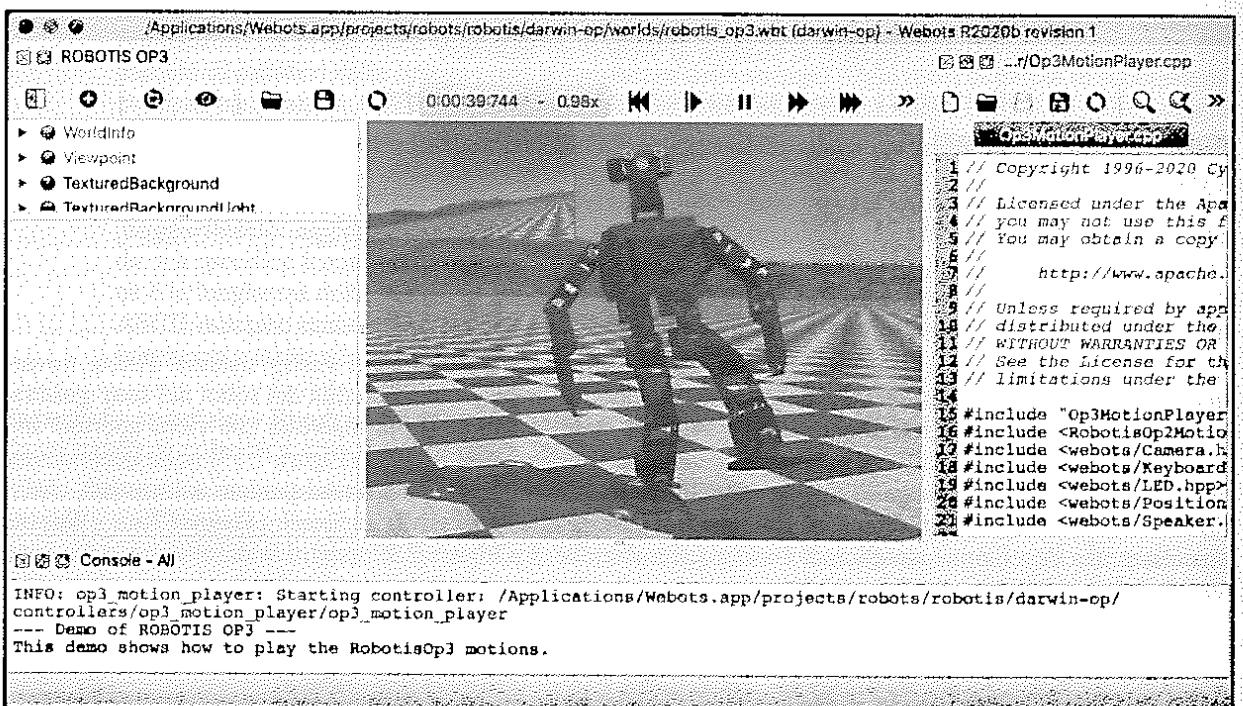


Figura 7.20 Una simulazione robotica di ROBOTIS OP3 eseguita in Webots.

Il tour guidato offre una panoramica di ciascuna simulazione, mostrata dal vivo nell'area di visualizzazione 3D dell'ambiente Webots (nella parte centrale della Figura 7.20). Come presto vedrete:

- la parte sinistra dell'ambiente vi consente di gestire e configurare i componenti della vostra simulazione robotica;
- la parte destra dell'ambiente fornisce un editor di codice integrato per scrivere e compilare il codice in C che controlla il robot nella vostra simulazione.

In questa schermata abbiamo ridotto le dimensioni dell'editor di codice. Come nella maggior parte degli IDE, si possono modificare le dimensioni delle aree all'interno della finestra trascinando le barre divisorie.

Panoramica dell'interfaccia utente

Per familiarizzare con l'interfaccia utente dell'ambiente Webots potete leggere la descrizione seguente:

<https://cyberbotics.com/doc/guide/the-user-interface>

Tutorial 1 di Webots: la vostra prima simulazione in Webots

Il team di Webots fornisce otto tutorial che introducono a molti aspetti dell'ambiente Webots e alle sue funzionalità per la simulazione di robot:

- Tutorial 1: la vostra prima simulazione in Webots
- Tutorial 2: modificare l'ambiente
- Tutorial 3: aspetto grafico
- Tutorial 4: approfondimenti sui controlli
- Tutorial 5: comporre solidi e caratteristiche fisiche
- Tutorial 6: robot a 4 ruote
- Tutorial 7: il vostro primo PROTO
- Tutorial 8: utilizzare il ROS

In questo caso pratico, seguirrete il primo tutorial di Webots per creare la simulazione di un robot usando diversi elementi predefiniti:

- una scacchiera (**RectangleArena**) nella quale si muoverà il vostro robot;
- diversi ostacoli in forma di scatole di legno (**WoodenBox**);
- un **robot e-puck**: un semplice robot simulato che si muove all'interno dell'area RectangleArena e cambia direzione quando incontra una WoodenBox o una parete.

Il simulatore e-puck corrisponde nel mondo reale a un robot educativo¹⁰ che ha:

- due ruote controllate in modo indipendente (note come **ruote differenziali**), in modo che il **robot può cambiare direzione** muovendo le ruote a differenti velocità;
- una telecamera (nell'area di visualizzazione 3D dell'ambiente Webots, nell'angolo a sinistra in alto, si trova una piccola finestra nella quale è possibile vedere **quello che "vede" un robot**);
- **otto sensori di distanza**;
- **10 luci a LED con intensità controllabile** (altri robot e-puck possono "vedere" queste luci tramite la loro telecamera permettendo così un'interazione visuale tra più robot e-puck).

Questo tutorial si concentra su come usare le ruote per far muovere il robot. Potete lavorare con i Tutorial 2-4 di Webots per usare ulteriori funzionalità del robot e-puck. Informazioni più approfondite su questo tipo di robot sono disponibili all'indirizzo:

<http://www.e-puck.org/>

10. "The e-puck, a Robot Designed for Education in Engineering." Accesso 13 dicembre 2020. <https://infoscience.epfl.ch/record/135236?ln=en>.

Fasi del tutorial

Trovate il vostro primo tutorial di Webots all'indirizzo

<https://cyberbotics.com/doc/guide/tutorial-1-your-first-simulation-in-webots>

Nei paragrafi seguenti esaminiamo ogni fase del tutorial, forniamo ulteriori approfondimenti e facciamo chiarezza su alcune istruzioni del tutorial.¹¹ I numeri delle fasi in questo esercizio corrispondono alle fasi “Hands-on” del tutorial di Webots. Per ciascuna fase, dovreste:

1. leggere il contenuto della fase nel tutorial di Webots;
2. leggere i nostri commenti aggiuntivi per quella fase;
3. eseguire le attività relative.

Il tutorial mostra soltanto un diagramma della simulazione robotica che creerete. Per aiutarvi, abbiamo aggiunto alcune schermate¹² che chiariscono le istruzioni del tutorial. **Nel corso delle varie fasi, assicuratevi di salvare ogni volta le modifiche effettuate. Se dovete resettare la simulazione, essa ritornerà all'ultima versione salvata.**

Fase 1: lancio dell'applicazione Webots

Nella Fase 1, aprirete semplicemente l'ambiente Webots. Esegirete quindi le fasi descritte qui di seguito.

Fase 2: creazione del vostro mondo virtuale

L'ambiente di simulazione è definito da un “mondo” che viene memorizzato in un file **.wbt**. Al suo interno, questo file utilizza il linguaggio per la simulazione della realtà virtuale **VRML** (*Virtual Reality Modeling Language*) per descrivere gli elementi del vostro mondo. Ogni mondo può avere caratteristiche, come la **gravità**, che influiscono sulle interazioni tra oggetti. Il mondo specifica in quale area il vostro robot può spostarsi e con quali oggetti può interagire. La procedura guidata **Create a Webots project directory** del menu **Wizards** (Figure 7.21-7.24) vi accompagnerà nella creazione di un nuovo mondo con la struttura di cartelle richiesta di Webots.

Nella prima schermata della procedura guidata **Create a Webots project directory** (Figura 7.21), fate clic su **Continue** (o **Next**) per passare alla fase successiva.

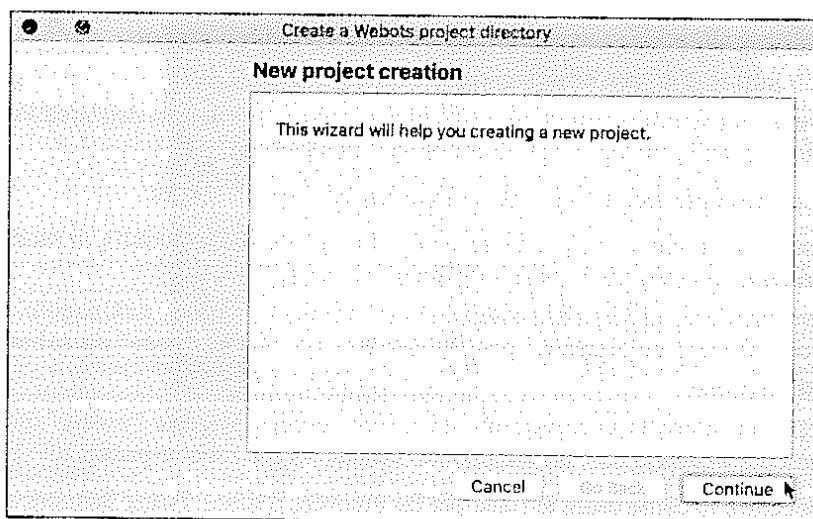


Figura 7.21 Finestra iniziale della procedura guidata **Create a Webots project directory**.

11. Questa parte è stata scritta nel dicembre 2020. Il software, la documentazione e i tutorial potrebbero essere cambiati. In caso di problemi, visitate il forum su Webots (<https://discord.com/invite/nTwBN9m>) e controllate le domande del Webots StackOverflow (<https://stackoverflow.com/questions/tagged/webots>) oppure inviateci una e-mail all'indirizzo deitel@deitel.com.

12. Abbiamo cambiato il colore dello sfondo nella nostra versione della simulazione per fare in modo che le schermate siano più leggibili.

Nella fase **Directory selection** della procedura guidata (Figura 7.22), modificate il nome della directory del vostro progetto da `my_project` (nome di default) a `my_first_simulation`; la posizione di default di questa cartella è la cartella `Documents` del vostro account utente.¹³

Nella fase **World settings** della procedura guidata (Figura 7.23), modificherete il nome file del mondo da `empty.wbt` (nome di default) a `my_first_simulation.wbt`, assicurandovi che tutte e quattro le caselle di controllo abbiano il segno di spunta. In questo modo al vostro mondo virtuale verranno aggiunti numerosi elementi predefiniti di Webots.

La fase **Conclusion** della procedura guidata (Figura 7.24) mostra tutte le cartelle e i file che verranno generati per la vostra simulazione. Nelle fasi successive, aggiungerete alcuni ostacoli e un robot e-puck, configurerete diverse impostazioni e scriverete del codice in C per controllare il robot.

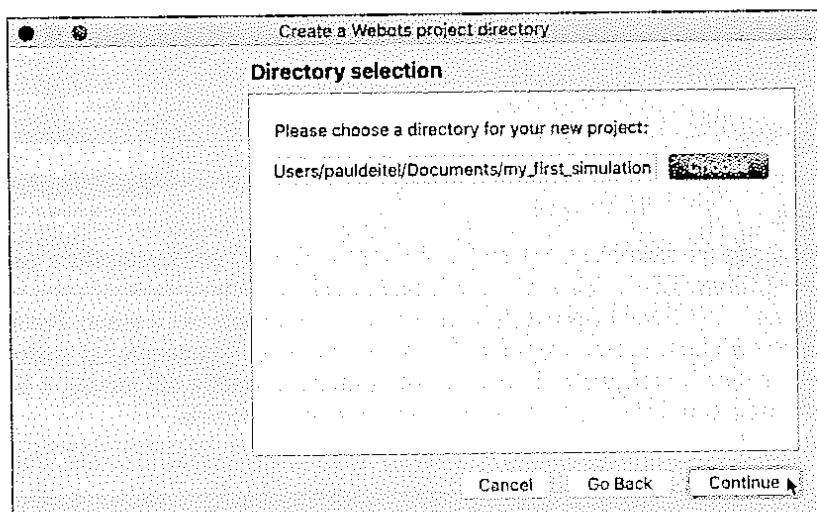


Figura 7.22 Modifica del nome di default della directory del progetto in `my_first_simulation`.

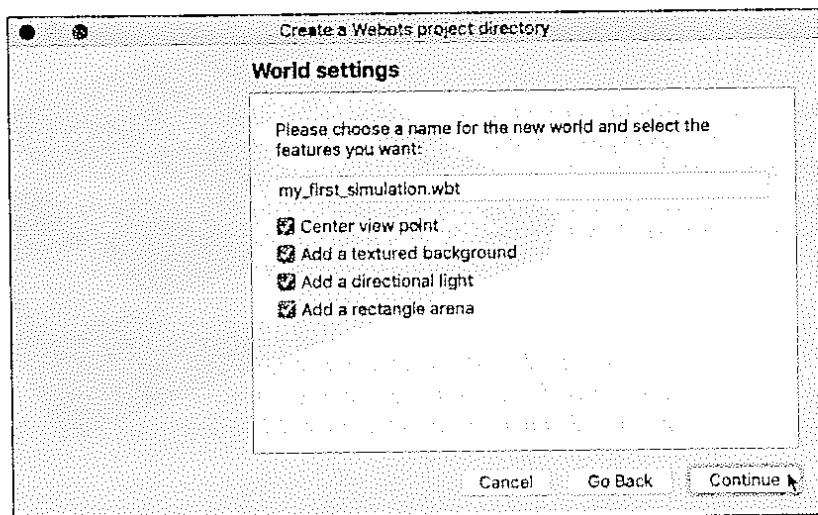


Figura 7.23 Modifica del nome di default del file del mondo e verifica della spunta delle caselle di controllo.

13. Potete modificare la posizione in cui salvare il vostro progetto Webots. Noi abbiamo scelto la cartella `/Users/pauldeitel/Documents` del nostro account utente, che vedrete in numerose schermate.

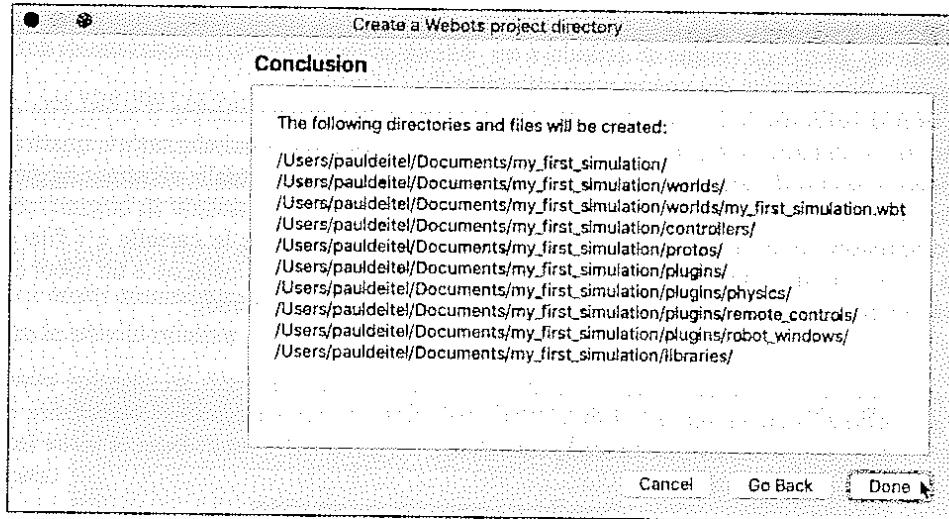


Figura 7.24 Elenco delle cartelle e dei file che verranno generati dalla procedura guidata per la simulazione.

Quando farete clic su **Done** (su macOS) o **Finish** (su Windows e Linux) nella procedura guidata **Create a Webots project directory** (Figura 7.24), l’area di visualizzazione 3D di Webots mostrerà una scacchiera RectangleArena vuota (Figura 7.25). Sono disponibili numerose opzioni di stile per il piano. Potete sperimentare con queste opzioni dopo aver imparato come cambiare le impostazioni degli elementi nel vostro mondo. Nell’area di visualizzazione 3D, potete ingrandire o ridurre la vista usando la rotella del mouse, e potete fare clic sulla RectangleArena e trascinarla per vederla da diverse angolazioni e ruotarla.

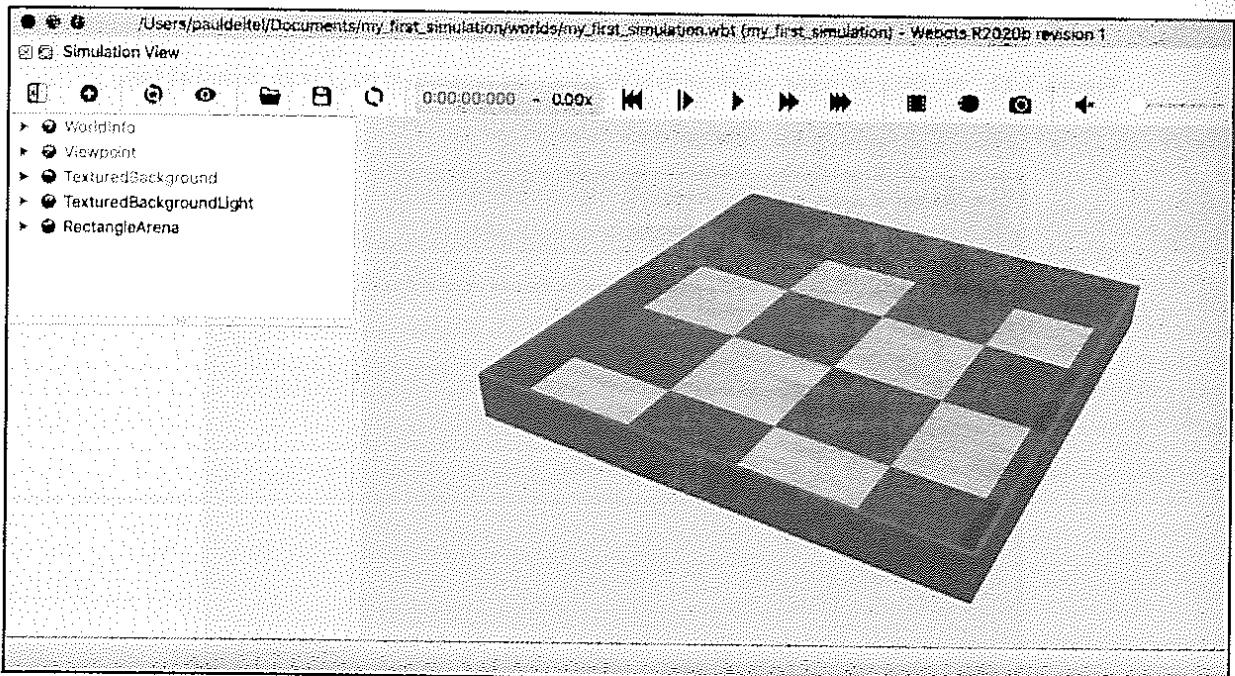


Figura 7.25 Vista iniziale della RectangleArena dopo il completamento della procedura guidata **Create a Webots project directory**.

Fase 3: modifica della RectangleArena

Ciascun elemento nella simulazione è un **nodo nell'albero della scena**, che potete vedere nella parte sinistra dell'ambiente Webots. In questa fase, selezionerete un nodo nell'albero della scena, poi ne modificherete alcune impostazioni, chiamate **campi**. Dopo che avrete eseguito questa fase, l'ambiente dovrebbe apparire come mostrato nella Figura 7.26.

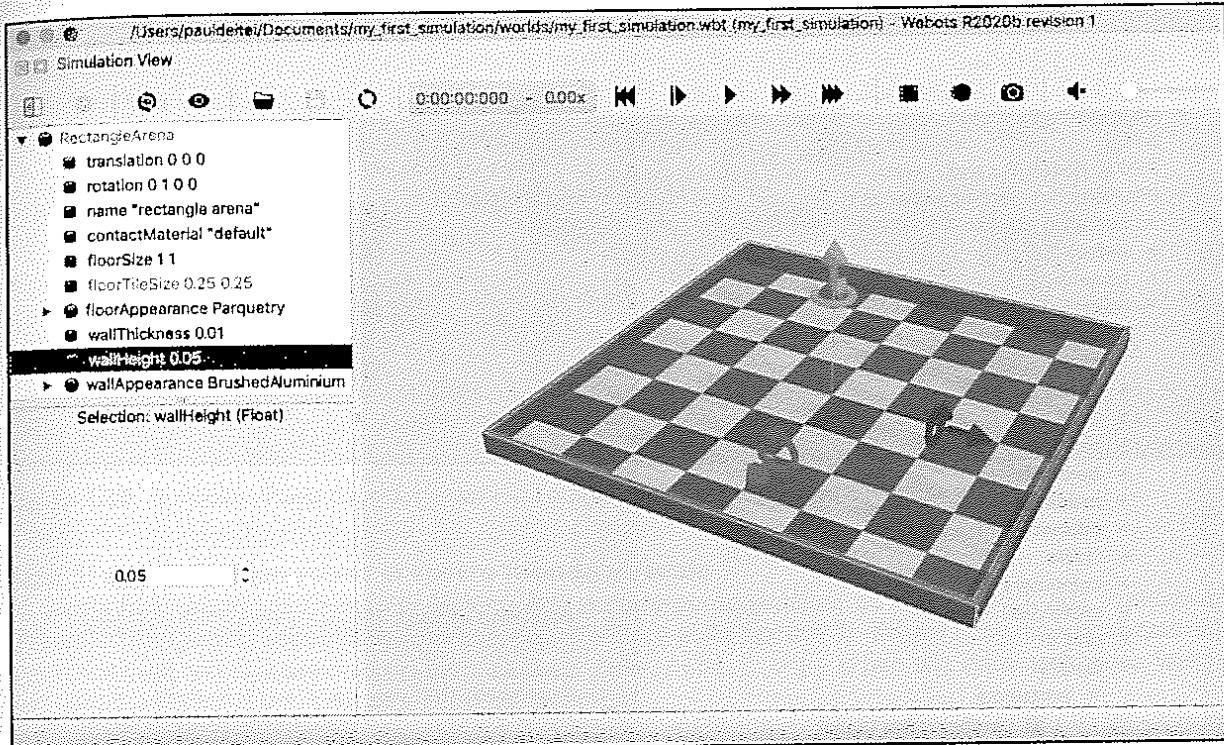


Figura 7.26 La RectangleArena dopo la modifica delle dimensioni dei quadri del piano a scacchi e la riduzione dell'altezza delle pareti.

Sopra la RectangleArena compaiono frecce colorate per ogni oggetto che avete selezionato nel mondo, facendo clic su di esso nell'area di visualizzazione 3D o facendo clic sul nodo relativo nell'albero della scena. Potete trascinare queste frecce per spostare, ruotare o inclinare l'oggetto.¹⁴ Trascinando una freccia dritta farete muovere l'oggetto lungo quell'asse; trascinando una freccia circolare farete ruotare o inclinare l'oggetto intorno a quell'asse. Eseguite un salvataggio del vostro ambiente, poi sperimentate con queste frecce per vedere come cambia la posizione della RectangleArena. Potete poi fare il reset dell'ambiente per tornare alla posizione originale.

Fase 4: aggiunta di ostacoli WoodenBox

L'ambiente Webots include quasi 800 oggetti e robot predefiniti, chiamati **nodi PROTO**. Un nodo PROTO descrive un oggetto complesso o un robot che potete aggiungere alle vostre simulazioni. L'ampia varietà di nodi PROTO consente di creare simulazioni 3D realistiche di ambienti del mondo reale. È possibile anche creare nodi PROTO personalizzati.

14. "Webots User Guide R2020b revision 2 – The 3D Window – Moving a Solid Object." Accesso 13 dicembre 2020, <https://cyberbotics.com/doc/guide/the-3d-window#moving-a-solid-object>.

In questa fase, aggiungerete un nodo PROTO WoodenBox utilizzando la finestra di dialogo **Add a node** (Figura 7.27). Quando selezionate un nodo PROTO nella finestra di dialogo, vengono visualizzati una breve descrizione del nodo, un link a una documentazione più dettagliata di tale nodo, e le informazioni relative alla sua licenza (con un link per ulteriori informazioni). Esplorate la finestra di dialogo **Add a node** per farvi un'idea della vasta quantità di nodi PROTO **animati** (robot e veicoli) e **inanimati** (pareti, edifici, arredi, piante ecc.) che potete usare nelle vostre simulazioni.

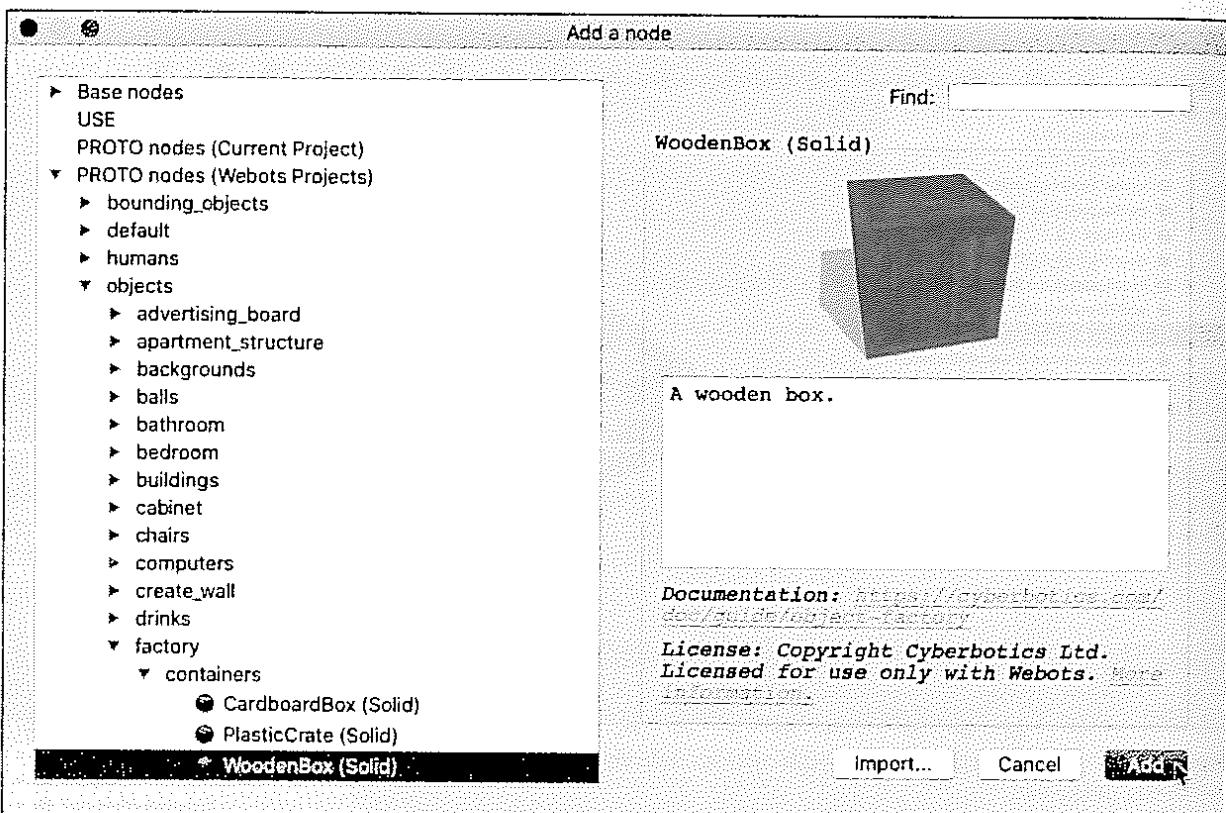


Figura 7.27 Selezione di un nodo PROTO WoodenBox nella finestra di dialogo **Add a node**.

Successivamente, **impostere le dimensioni della WoodenBox e la sposterete**. Quindi ne farete **due copie** e le sposterete in altri punti della RectangleArena. Quando in questa fase vi viene richiesto di copiare e incollare una WoodenBox, la copia avrà la stessa dimensione e la stessa posizione dell'originale. **Tenete premuto il tasto Maiusc** e trascinate la nuova copia in una posizione differente per poterla vedere. Abbiamo trovato più facile copiare i nodi selezionandoli nell'albero della scena. Quando avrete completato questa fase, il vostro mondo dovrebbe essere simile a quanto mostrato nella Figura 7.28. L'ultima WoodenBox che avete mosso sarà selezionata nel vostro mondo.¹⁵

15. "Webots User Guide R2020b revision 2 – The 3D Window." Accesso 13 dicembre 2020. <https://cyberbotics.com/doc/guide/the-3d-window#moving-a-solid-object>.

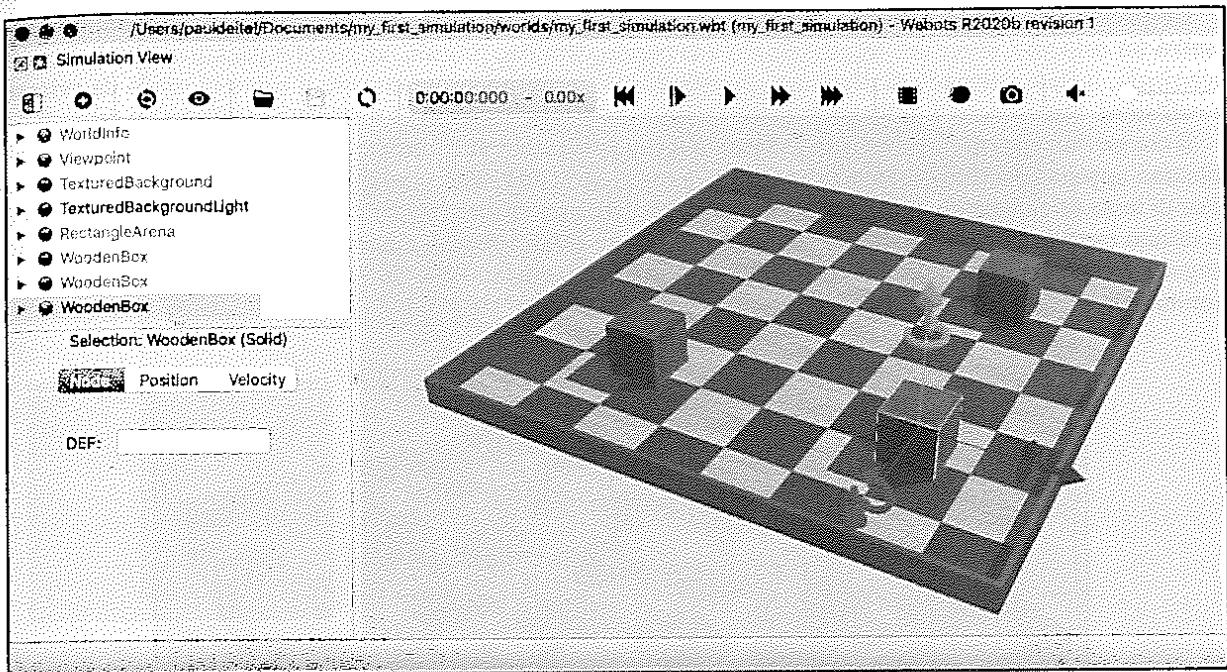


Figura 7.28 Il mondo virtuale dopo la creazione e il posizionamento di tre oggetti WoodenBox.

Fase 5: aggiunta di un robot al vostro mondo virtuale

In questa fase, userete la finestra di dialogo **Add a node** per aggiungere un robot e-puck alla simulazione (Figura 7.29). Quando selezionate il nodo E-puck (Robot), nella finestra di dialogo vengono visualizzati una descrizione del robot, il sito web del robot (<http://www.e-puck.org>), il link alla documentazione Webots del robot e le informazioni relative alla licenza del robot.

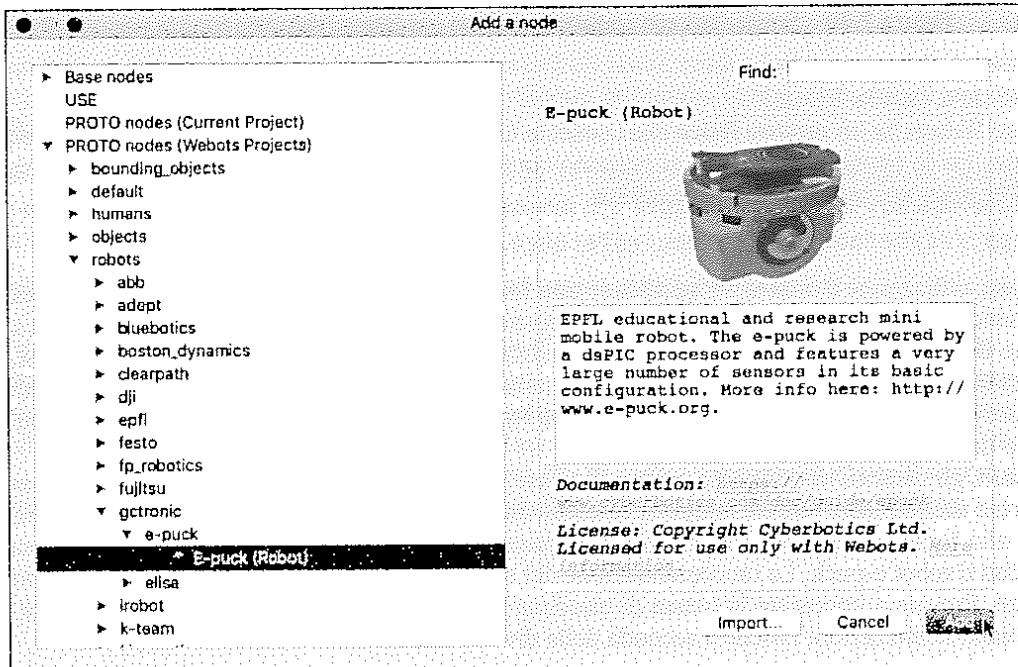


Figura 7.29 Aggiunta di un nodo PROTO di tipo robot e-puck alla simulazione.

L'e-puck è preconfigurato per spostarsi in avanti, ruotando a sinistra per cambiare direzione se **contra un ostacolo**, come una WoodenBox o una parete. Sebbene piccolo, un robot e-puck in realtà è ricco di tecnologia, inclusi i **sensori di distanza**, che si possono usare per programmare il robot in modo da **evitare le collisioni**. Nel Tutorial 4 di Webots, apprenderete come usare i sensori di distanza per evitare gli ostacoli.

Il comportamento di un robot è specificato dal suo **controller**. Il controller di default del robot e-puck che abbiamo appena descritto è chiamato **e-puck_avoid_obstacles** (è possibile vedere questo codice nell'editor di testo selezionando **Tools > Text Editor**). Studiare i controller esistenti inclusi nell'ambiente Webots è un ottimo metodo per saperne di più su come controllare i suoi robot. A completamento di questa fase, farete eseguire il controller di default del robot (Figura 7.30). Abbiamo fatto clic sull'area dello sfondo del mondo per eliminare la selezione da tutti gli elementi della simulazione.

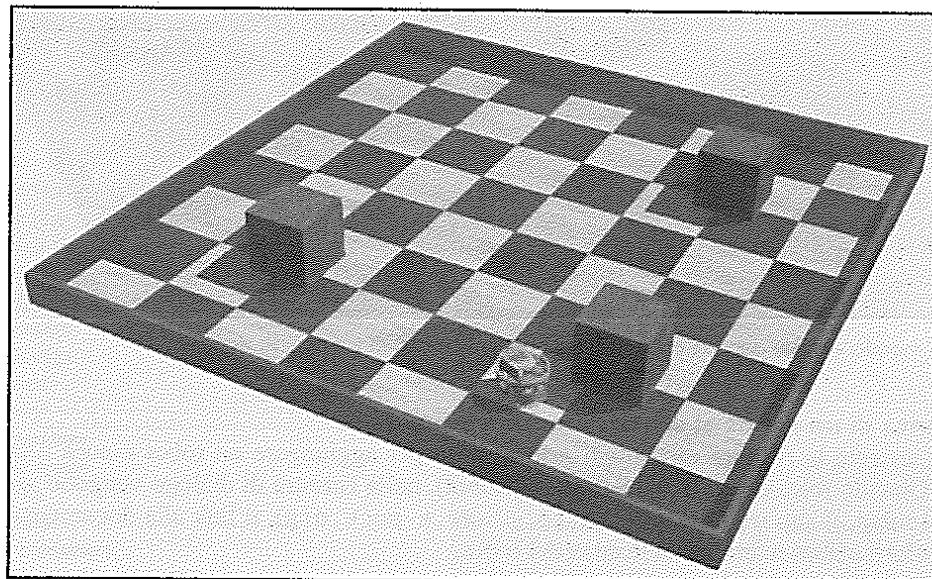


Figura 7.30 Simulazione con un robot e-puck in movimento nella RectangleArena.

Fase 6: giocare con la fisica

Il simulatore Webots ha un **motore fisico** che consente di far agire e interagire gli oggetti come accadrebbe nel mondo reale. Tra le opzioni fisiche che si possono configurare sono incluse densità, massa, inerzia, attrito e rimbalzo. Per maggiori informazioni, consultate:

<https://cyberbotics.com/doc/reference/physics>

In questa fase, utilizzerete il mouse per **applicare una forza al robot e-puck**. Una volta eseguita questa azione, il vostro mondo dovrebbe essere simile a quanto mostrato nella Figura 7.31, con la freccia rossa che indica la direzione della forza. Nel corso di questa fase, potrete inadvertitamente far ribaltare il robot se applicate troppa forza (come mostrato nella Figura 7.31). Ovviamente, i robot possono ribaltarsi anche nel mondo reale. Per rimediare nella simulazione, fate clic sul pulsante **Reset Simulation**, che riporterà la simulazione com'era al momento dell'ultimo salvataggio.

Per default, gli ostacoli WoodenBox creati nella *Fase 4* sono fissi sulla scacchiera e non si muovono quando vengono urtati dall'e-puck. Vedrete in questa fase che impostando la massa delle WoodenBox potrete far sì che reagiscano alla forza spostandosi. Più piccola è la massa della WoodenBox, più si sposterà al momento della collisione con il robot e-puck. La raccomandazione del tutorial è di impostare la massa delle WoodenBox a 0,2 chilogrammi. Provate a impostare la massa di ciascuna WoodenBox a valori minori e maggiori per osservare come cambiano le interazioni fisiche a seconda delle diverse masse.

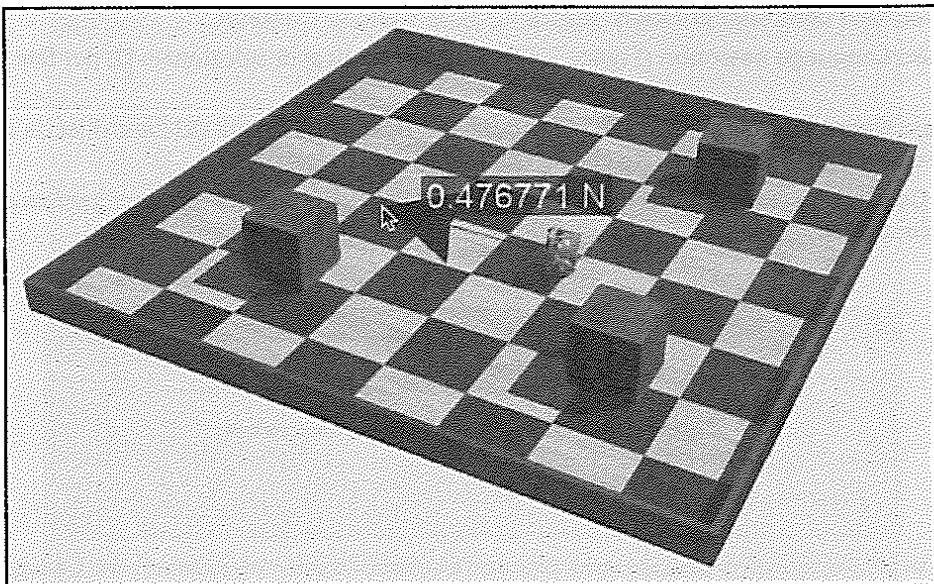


Figura 7.31 Applicazione manuale di una forza al robot e-puck. In questo caso è stata applicata troppa forza, causando il ribaltamento del robot.

Fase 7: diminuzione del passo temporale del mondo

Nel corso di una simulazione, Webots tiene traccia del **tempo virtuale** della simulazione. Il **passo temporale di base** è un valore in millisecondi. Per tutta la durata della simulazione, quando il tempo virtuale aumenta secondo il passo temporale di base, Webots effettua i suoi calcoli fisici.¹⁶

- Valori maggiori del passo temporale di base diminuiscono la frequenza con cui vengono effettuati i calcoli fisici. Ciò permette un'esecuzione più rapida delle simulazioni perché svolgono un numero minore di calcoli; tuttavia, le interazioni fisiche, come le collisioni tra oggetti, potrebbero risultare meno precise, e la simulazione quindi potrebbe sembrare poco armonica.
- Valori minori aumentano invece la frequenza dei calcoli fisici, rendendoli più accurati. Le simulazioni quindi vengono eseguite più lentamente perché svolgono un numero maggiore di calcoli, ma i movimenti risultano più fluidi.

Quando avete creato i file per questa simulazione, Webots ha impostato il passo temporale di base a 32 millisecondi. In questa fase, lo diminuirete fino a 16 millisecondi. Per suggerimenti su velocità e prestazioni delle simulazioni di Webots, consultate:

<https://www.cyberbotics.com/doc/guide/speed-performance>

Fase 8: creazione di un file di codice sorgente in C per il controller del vostro robot

In questa fase, sostituirrete il controller di default `e-puck_avoid_obstacles` con un nuovo **controller personalizzato**. Ogni controller da voi creato può essere usato da molti robot, ma ciascun robot può avere un solo controller alla volta. Le Figure 7.32-7.35 mostrano le fasi successive alla selezione di **Wizards > New Robot Controller....** Queste fasi vi aiuteranno a **creare un nuovo file di codice sorgente in C per il vostro controller personalizzato** e di aprirlo nell'editor di codice situato nella parte destra dell'ambiente Webots:

16. "Webots Reference Manual R2020b revision 2: WorldInfo." Accesso 12 dicembre 2020. <https://www.cyberbotics.com/doc/reference/worldinfo>.

- nella fase **New controller creation** (Figura 7.32), fate clic su **Continue**;
- nella fase **Language selection** (Figura 7.33), assicuratevi di aver selezionato **C**, poi fate clic su **Continue**;
- nella fase **Name selection** (Figura 7.34), modificate il nome di default del controller da **my_controller** in **epuck_go_forward** e fate clic su **Continue**;
- la fase **Conclusion** (Figura 7.35) mostra tutte le cartelle e i file che la procedura guidata genererà per il vostro controller.

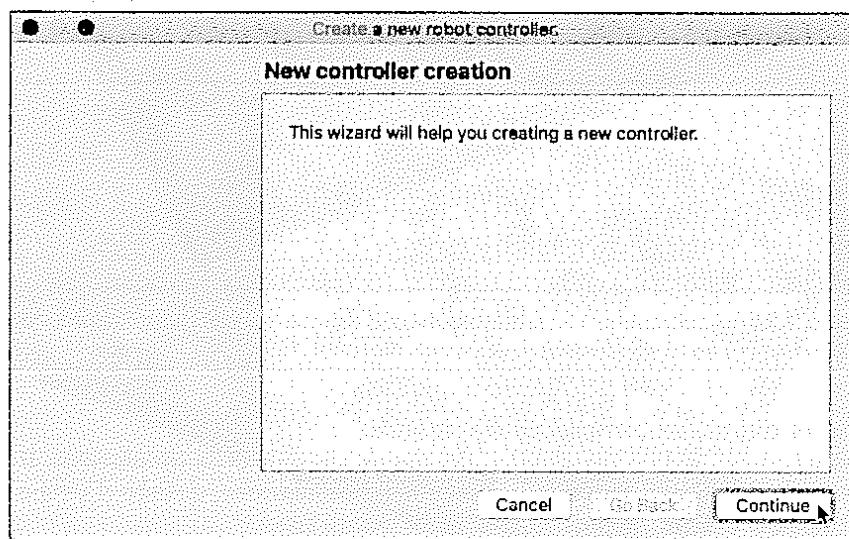


Figura 7.32 Finestra iniziale della procedura guidata **Create a new robot controller**.

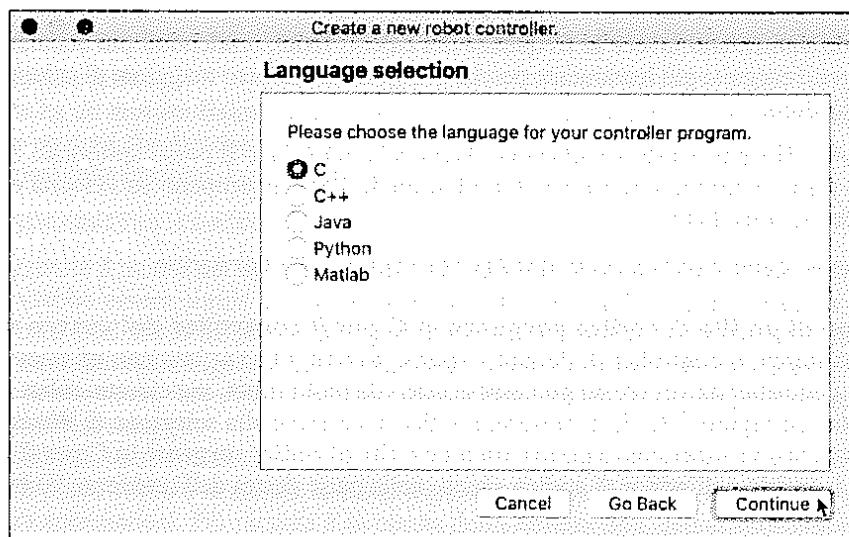


Figura 7.33 Selezione del linguaggio di programmazione C per il controller personalizzato del vostro robot.

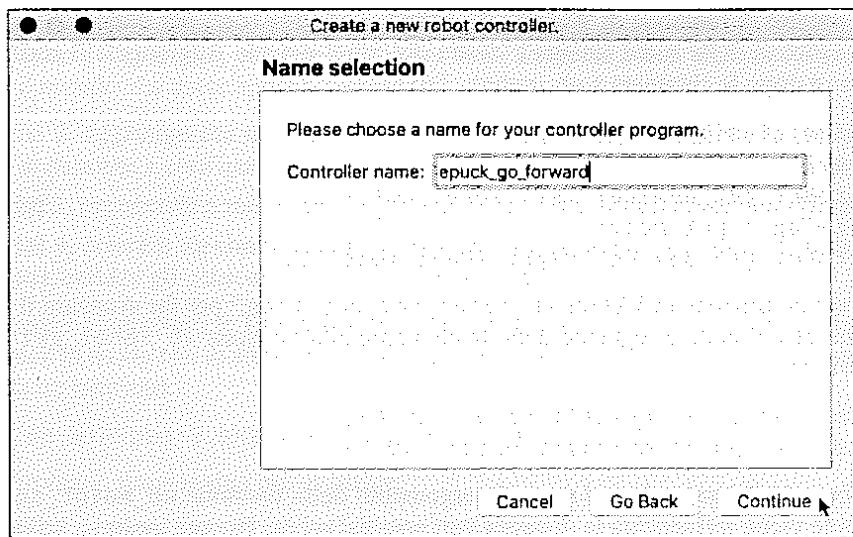


Figura 7.34 Modifica del nome di default del controller personalizzato in epuck_go_forward.

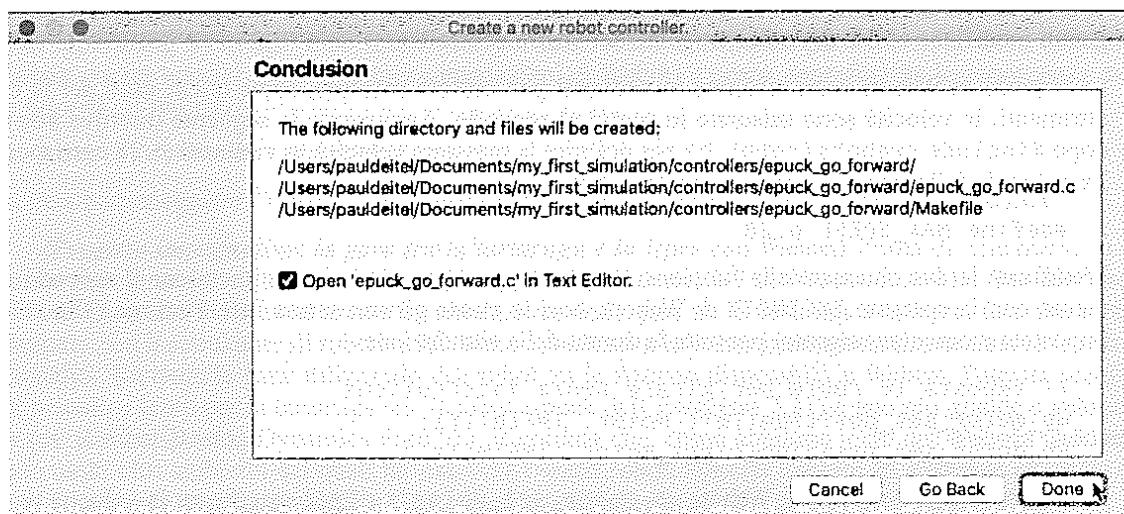


Figura 7.35 Riepilogo delle cartelle e dei file che la procedura guidata genererà per il vostro controller.

Fase 9: modifica del codice del vostro controller per far muovere in avanti il robot

Il nuovo controller creato nella *Fase 8* contiene la struttura di base di un semplice controller. Ora aggiungerete a questo file il codice che farà muovere in avanti il robot e-puck per un breve tratto. Il tutorial non specifica esattamente dove inserire ogni nuova istruzione nel codice del controller, pertanto effettuate le seguenti modifiche al file di codice sorgente del vostro controller.

1. Aggiungete la seguente direttiva #include prima della funzione main:

```
#include <webots/motor.h>
```

2. In seguito, userete la funzione di Webots `wb_robot_get_device`¹⁷ per ottenere i dispositivi che rappresentano i motori delle ruote sinistra e destra del robot e-puck. Aggiungete il codice seguente nella funzione `main` dopo la chiamata a `wb_robot_init`:

```
// ottieni i dispositivi motore
WbDeviceTag left_motor =
    wb_robot_get_device("left wheel motor");
WbDeviceTag right_motor =
    wb_robot_get_device("right wheel motor");
```

3. Infine, userete la funzione di Webots `wb_motor_set_position`¹⁸ per far avanzare il robot di un breve tratto. Aggiungete il codice seguente dopo le chiamate alla funzione `wb_robot_get_device` e prima del ciclo `while`:

```
wb_motor_set_position(left_motor, 10.0);
wb_motor_set_position(right_motor, 10.0);
```

Sperimentate attribuendo diversi valori al secondo argomento della funzione `wb_motor_set_position` per farvi un'idea di come viene influenzata la distanza percorsa dal robot. Provate inoltre a utilizzare valori differenti nelle due chiamate a `wb_motor_set_position` in modo che la rotazione di ciascuna ruota sia diversa.

Fase 10: modifica del codice del vostro controller per cambiare la velocità del robot

In questa fase finale, modificherete il codice del vostro controller specificando la velocità delle ruote. Effettuate le seguenti modifiche al file di codice sorgente del vostro controller.

1. Le velocità in Webots sono misurate in **radiani per motori rotatori**, come quelli usati per le ruote; altrimenti, le velocità sono misurate in **metri al secondo**. Aggiungete la seguente direttiva `#define` dopo `#include <webots/robot.h>` per definire la massima **velocità in radienti** (6.28 corrisponde a 2π radianti) della rotazione delle ruote del robot:

```
#define MAX_SPEED 6.28
```

2. Modificate le due chiamate alla funzione `wb_motor_set_position`, sostituendone il secondo argomento con la costante **INFINITY** di Webots, così le **ruote girano senza interruzione** (alla velocità impostata momentaneamente) per tutta la durata della simulazione:

```
// imposta le ruote per girare senza interruzione
wb_motor_set_position(left_motor, INFINITY);
wb_motor_set_position(right_motor, INFINITY);
```

3. Infine, userete la funzione di Webots `wb_motor_set_velocity`¹⁹ per specificare la velocità di rotazione delle ruote, in radiani al secondo. Aggiungete il codice seguente dopo le chiamate alla funzione `wb_robot_get_device` e prima del ciclo `while`:

```
// imposta la velocità al 10% di MAX_SPEED
wb_motor_set_velocity(left_motor, 0.1 * MAX_SPEED);
wb_motor_set_velocity(right_motor, 0.1 * MAX_SPEED);
```

Sperimentate attribuendo diversi valori al secondo argomento della funzione `wb_motor_set_velocity` per farvi un'idea di come viene influenzata la velocità del robot. Provate a utilizzare valori differenti nelle due chiamate in modo che le due ruote non girino all'unisono. Prestate attenzione a come impostate questi valori: potrebbero far girare in tondo il robot.

17. "Webots Reference Manual R2020b revision 2 – Robot." Accesso 13 dicembre 2020. https://cyberbotics.com/doc/reference/robot#wb_robot_get_device.

18. "Webots Reference Manual R2020b revision 2 – Motor." Accesso 13 dicembre 2020. https://cyberbotics.com/doc/reference/motor#wb_motor_set_position.

19. "Webots Reference Manual R2020b revision 2 – Motor." Accesso 13 dicembre 2020. https://cyberbotics.com/doc/reference/motor#wb_motor_set_velocity.

Ulteriori tutorial

Dopo aver completato il **Tutorial 1**, potrete continuare con i **Tutorial 2-8** di Webots. I **Tutorial 2-4** effettuano varie modifiche al mondo che avete appena creato:

- nel **Tutorial 2**, aggiungerete una palla nel vostro mondo. Approfondirete la conoscenza dei vari tipi di nodo e imparerete a configurare le opzioni fisiche che fanno rotolare la palla nella simulazione;
- nel **Tutorial 3**, imparerete a migliorare la grafica della vostra simulazione con effetti di luce e texture;
- nel **Tutorial 4**, creerete un controller più sofisticato che consente all'e-puck di usare i suoi sensori di distanza per evitare gli ostacoli da voi creati in precedenza.

Sfida: Quando avrete completato il **Tutorial 4**, provate a costruire nel vostro mondo un labirinto e programmare il robot e-puck in modo che lo attraversi. Nella seguente pagina web sono elencati numerosi algoritmi per trovare l'uscita da un labirinto:

https://en.wikipedia.org/wiki/Maze_solving_algorithm

I **Tutorial 5-7** affrontano funzionalità più avanzate:

- nel **Tutorial 5**, apprenderete di più sulla fisica in Webots;
- nel **Tutorial 6**, lavorerete con un **robot a quattro ruote** e imparerete di più sui **sensori**;
- nel **Tutorial 7**, sarete guidati nella **creazione del vostro nodo PROTO**.

Nel **Tutorial 8**, di livello avanzato, imparerete a lavorare con i nodi Webots inclusi in `webots_ros`. Il **ROS (Robot Operating System)**²⁰ è un framework per la scrittura di software per i robot. Se avete intenzione di affrontare questo tutorial, seguite le raccomandazioni di Webots e prima approfondite la conoscenza del ROS con i tutorial all'indirizzo

<http://wiki.ros.org/ROS/Tutorials>

Riuscire a padroneggiare i Tutorial 2-8 di Webots è un risultato degno di menzione sul curriculum.

7.32 (Progetto impegnativo: la gara tra la tartaruga e la lepre con Webots) Prima di cimentarvi in questo esercizio impegnativo, vi raccomandiamo di completare i Tutorial 2-7 di Webots citati alla fine dell'Esercizio 7.31. Nell'Esercizio 5.54, avete creato la simulazione della gara tra la tartaruga e la lepre. Ora che avete fatto conoscenza con il simulatore di robotica 3D Webots, scatenate la vostra immaginazione usandone le incredibili funzionalità. Ricreate la gara utilizzando due robot tra le dozzine disponibili in Webots. Potrete scegliere un robot piccolo e lento per la tartaruga (es. il robot e-puck dell'Esercizio 7.31) e uno più grande e veloce per la lepre (es. il robot Boston Dynamics Spot^{21,22}). Ricordate che, come abbiamo visto nel **Webots Guided Tour**, ci sono molti altri ambienti nei quali far muovere i vostri robot. Potrete copiare un ambiente esistente con un terreno contenente oggetti quali erba, fiori, alberi e colline nel quale far gareggiare i vostri robot.

20. "About ROS." Accesso 13 dicembre 2020. <https://www.ros.org/about-ros/>.

21. "Webots User Guide ~ Boston Dynamics' Spot." Accesso 31 dicembre 2020. <https://www.cyberbotics.com/doc/guide/spot>.

22. "Spot." Accesso 31 dicembre 2020. <https://www.bostondynamics.com/spot>.

CAPITOLO

8

Sommario del capitolo

- 8.1 Introduzione
- 8.2 Nozioni fondamentali su stringhe e caratteri
- 8.3 Libreria per il trattamento dei caratteri
- 8.4 Funzioni di conversione di stringhe
- 8.5 Funzioni della libreria standard di input/output
- 8.6 Funzioni per la manipolazione di stringhe della libreria per il trattamento delle stringhe
- 8.7 Funzioni di confronto della libreria per il trattamento delle stringhe
- 8.8 Funzioni per la ricerca della libreria per il trattamento delle stringhe
- 8.9 Funzioni di gestione della memoria della libreria per il trattamento delle stringhe
- 8.10 Altre funzioni della libreria per il trattamento delle stringhe
- 8.11 Programmazione sicura in C
- 8.12 Riepilogo

Caratteri e stringhe

Obiettivi

- Usare le funzioni della libreria per il trattamento dei caratteri (`<ctype.h>`).
- Usare le funzioni di conversione di stringhe della libreria di utilità generale (`<stdlib.h>`).
- Usare le funzioni di input/output di stringhe e caratteri della libreria standard di input/output (`<stdio.h>`).
- Usare le funzioni di elaborazione delle stringhe della libreria per il trattamento delle stringhe (`<string.h>`).
- Usare le funzioni di gestione della memoria della libreria per il trattamento delle stringhe (`<string.h>`).

8.1 Introduzione

Il presente capitolo introduce le funzioni della Libreria Standard del C che facilitano l'elaborazione di caratteri, stringhe, righe di testo e blocchi di memoria. Il capitolo prende in esame le tecniche utilizzate per sviluppare editor, word processor, software per l'impaginazione e altri tipi di software per l'elaborazione del testo. È possibile implementare le manipolazioni di testi eseguite da funzioni per la formattazione dell'input/output come `printf` e `scanf` usando le funzioni esaminate in questo capitolo.

8.2 Nozioni fondamentali su stringhe e caratteri

I caratteri sono i blocchi costituenti fondamentali dei programmi. Ogni programma è composto da una sequenza di caratteri che, quando sono raggruppati insieme in modo sensato, è interpretata dal computer come una serie di istruzioni usate per eseguire un compito. Un programma può contenere **costanti carattere**, ciascuna delle quali è un valore `int` rappresentato come un carattere tra virgolette singole. Il valore di una costante carattere è il valore intero del carattere nell'**insieme di caratteri** della macchina. Per esempio, 'z' rappresenta il valore intero della lettera z, e '\n' rappresenta il valore intero del carattere newline.

Una **stringa** è una serie di caratteri trattati come unità singola. Una stringa può comprendere lettere, cifre e vari **caratteri speciali** come +, -, *, / e \$. Le **stringhe letterali**, o **stringhe costanti**, sono scritte tra virgolette doppie come segue:

"John Q. Doe"	(un nome)
"99999 Main Street"	(l'indirizzo di una via)
"Waltham, Massachusetts"	(una città e uno stato)
"(201) 555-1212"	(un numero di telefono)

Le stringhe terminano con il carattere nullo

 Tutte le stringhe devono finire con il **carattere nullo** ('\0'). Stampare una "stringa" che non contiene un carattere nullo di terminazione è un errore logico, che porta a risultati di questo sono indefiniti. Su alcuni sistemi, la stampa continuerà dopo la fine della "stringa" finché non verrà incontrato un carattere nullo. Su altri sistemi, il programma terminerà prima del dovuto (ovvero si arresterà) segnalando un "errore di segmentazione" o una "violazione di accesso".

Stringhe e puntatori

Si accede a una stringa mediante un *puntatore* al suo primo carattere. Il "valore" di una stringa è l'indirizzo del suo primo carattere. Pertanto, in C, è appropriato dire che una stringa è un puntatore al primo carattere della stringa. Proprio come gli array, in quanto le stringhe sono semplicemente array di caratteri.

Inizializzazione di array di tipo char e puntatori di tipo char *

È possibile inizializzare con una stringa un array di caratteri o una variabile di tipo char *. Le definizioni

```
char color[] = "blue";
const char *colorPtr = "blue";
```

inizializzano le variabili color e colorPtr con la stringa "blue". La prima definizione crea un array color di 5 elementi che contengono i caratteri *modificabili* 'b', 'l', 'u', 'e' e '\0'. La seconda definizione crea la variabile puntatore colorPtr che punta alla lettera 'b' nella stringa "blue", che è *non modificabile*.

La definizione dell'array color può anche essere scritta come

```
char color[] = {'b', 'l', 'u', 'e', '\0'};
```

La definizione precedente determina automaticamente la dimensione dell'array in base al suo numero di inizializzatori (5). Quando si memorizza una stringa in un array di tipo char, l'array deve essere abbastanza

 grande da contenere la stringa e il suo carattere nullo di terminazione. Non allocare in un array di caratteri spazio sufficiente per memorizzare il carattere nullo che termina una stringa è un errore. Il C permette la memorizzazione di stringhe di qualsiasi lunghezza. Se una stringa è più lunga dell'array di tipo char nella quale viene memorizzata, i caratteri oltre la fine dell'array potrebbero sovrascrivere altri dati in memoria.

Le stringhe letterali non devono essere modificate

 Il C standard indica che una stringa letterale è immutabile (cioè, non modificabile). Se avete necessità di modificare una stringa letterale, questa deve essere memorizzata in un array di caratteri.

Lettura di una stringa con la funzione scanf

La funzione scanf può leggere una stringa e memorizzarla in un array di caratteri. Assumiamo di avere un array di caratteri word contenente 20 elementi. Si può leggere una stringa e memorizzarla in un array con l'istruzione

```
scanf("%19s", word);
```

Poiché word è un array, il nome dell'array è un puntatore al primo elemento dell'array. Pertanto, non è necessaria la & che viene usata normalmente con gli argomenti di scanf.

Ricordate dal Paragrafo 6.5.4 che la funzione scanf legge i caratteri finché non incontra uno spazio, una tabulazione, un indicatore di nuova riga o l'indicatore di end-of-file. L'ampiezza di campo 19 nell'istruzione precedente garantisce che scanf legga un *massimo* di 19 caratteri, conservando l'ultimo carattere per il carattere nullo di terminazione della stringa. Questo evita che scanf scriva in memoria caratteri oltre l'ultimo elemento dell'array.

Senza l'ampiezza di campo 19 nella specifica di conversione %19s, l'input dell'utente potrebbe superare i 19 caratteri e sovrascrivere altri dati in memoria. In questo caso il vostro programma potrebbe arrestarsi oppure sovrascrivere altri dati in memoria. Perciò, indicate sempre un'ampiezza di campo quando usate scanf per leggere le stringhe. (Per leggere righe di input di lunghezza arbitraria si può usare la funzione readline non standard ma ampiamente compatibile, solitamente inclusa in stdio.h).

✓ Autovalutazione

1. (*Completare*) Si accede a una stringa mediante un _____ al suo primo carattere.

Risposta: puntatore.

2. (*Vero/Falso*) La definizione seguente inizializza l'array color con la stringa di caratteri "blue":

```
char color[] = {'b', 'l', 'u', 'e'};
```

Risposta: *Falso*. In realtà, per essere una stringa di caratteri, l'array color deve finire con il carattere nullo, come in

```
char color[] = {'b', 'l', 'u', 'e', '\0'};
```

3. (*Vero/Falso*) Stampare una "stringa" che non contiene un carattere nullo di terminazione è un errore logico: l'esecuzione del programma termina immediatamente.

Risposta: *Falso*. In realtà, la stampa continuerà dopo la fine della "stringa" finché non verrà incontrato un carattere nullo.

8.3 Libreria per il trattamento dei caratteri

La libreria per il trattamento dei caratteri (<cctype.h>) include diverse funzioni che eseguono test e manipolazioni di dati di tipo carattere. Ogni funzione riceve come argomento un `unsigned char` (rappresentato come un `int`) o EOF. Come abbiamo visto nel Capitolo 4, i caratteri sono spesso manipolati come interi, perché un carattere in C è un intero di un byte. EOF normalmente ha il valore -1. La tabella seguente riepiloga le funzioni della libreria per il trattamento dei caratteri.

Prototipo	Descrizione della funzione
<code>int isblank(int c);</code>	Restituisce un valore vero se c è un carattere vuoto che separa le parole in una riga di testo e 0 (falso) altrimenti.
<code>int isdigit(int c);</code>	Restituisce un valore vero se c è una cifra e 0 (falso) altrimenti.
<code>int isalpha(int c);</code>	Restituisce un valore vero se c è una lettera e 0 (falso) altrimenti.
<code>int isalnum(int c);</code>	Restituisce un valore vero se c è una cifra o una lettera e 0 (falso) altrimenti.
<code>int isxdigit(int c);</code>	Restituisce un valore vero se c è una cifra esadecimale e 0 (falso) altrimenti. (Si veda l'Appendice E, disponibile sulla piattaforma MyLab, per una descrizione dettagliata di numeri binari, numeri ottali, numeri decimali e numeri esadecimali.)
<code>int islower(int c);</code>	Restituisce un valore vero se c è una lettera minuscola e 0 (falso) altrimenti.
<code>int isupper(int c);</code>	Restituisce un valore vero se c è una lettera maiuscola e 0 (falso) altrimenti.
<code>int tolower(int c);</code>	Se c è una lettera maiuscola, tolower restituisce c come lettera minuscola; altrimenti, restituisce l'argomento inalterato.
<code>int toupper(int c);</code>	Se c è una lettera minuscola, toupper restituisce c come lettera maiuscola; altrimenti, restituisce l'argomento inalterato.



► <code>int isspace(int c);</code>	Restituisce un valore vero se c è un carattere di spaziatura – newline ('\n'), spazio (' '), avanzamento pagina ('\f'), ritorno a capo ('\r'), tab orizzontale ('\t') o tab verticale ('\v') – altrimenti, restituisce 0 (falso).
<code>int iscntrl(int c);</code>	Restituisce un valore vero se c è un carattere di controllo – tab orizzontale ('\t'), tab verticale ('\v'), avanzamento pagina ('\f'), avviso ('\a'), backspace ('\b'), ritorno a capo ('\r'), newline ('\n') e altri – altrimenti, restituisce 0 (falso).
<code>int ispunct(int c);</code>	Restituisce un valore vero se c è un carattere stampabile diverso da uno spazio, da una cifra o da una lettera – come \$, #, (,), [,], {, }, :, ; o % – altrimenti, restituisce 0 (falso).
<code>int isprint(int c);</code>	Restituisce un valore vero se c è un carattere stampabile (cioè un carattere visibile sullo schermo) incluso uno spazio; altrimenti, restituisce 0 (falso).
<code>int isgraph(int c);</code>	Restituisce un valore vero se c è un carattere stampabile diverso da uno spazio; altrimenti, restituisce 0 (falso).

8.3.1 Funzioni `isdigit`, `isalpha`, `isalnum` e `isxdigit`

Il programma della Figura 8.1 illustra le funzioni `isdigit`, `isalpha`, `isalnum` e `isxdigit`. La funzione `isdigit` determina se il suo argomento è una cifra (0-9). La funzione `isalpha` determina se il suo argomento è una lettera maiuscola (A-Z) o una lettera minuscola (a-z). La funzione `isalnum` determina se il suo argomento è una lettera maiuscola, una lettera minuscola o una cifra. La funzione `isxdigit` determina se il suo argomento è una **cifra esadecimale** (A-F, a-f, 0-9).

```

1 // fig08_01.c
2 // Uso delle funzioni isdigit, isalpha, isalnum e isxdigit
3 #include <ctype.h>
4 #include <stdio.h>
5
6 int main(void) {
7     printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
8         isdigit('8') ? "8 is a " : "8 is not a ", "digit",
9         isdigit('#') ? "# is a " : "# is not a ", "digit");
10
11    printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalpha: ",
12        isalpha('A') ? "A is a " : "A is not a ", "letter",
13        isalpha('b') ? "b is a " : "b is not a ", "letter",
14        isalpha('&') ? "& is a " : "& is not a ", "letter",
15        isalpha('4') ? "4 is a " : "4 is not a ", "letter");
16
17    printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalnum:",
18        isalnum('A') ? "A is a " : "A is not a ", "digit or a letter",
19        isalnum('8') ? "8 is a " : "8 is not a ", "digit or a letter",
20        isalnum('#') ? "# is a " : "# is not a ", "digit or a letter");
21
22    printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to isxdigit:",
23        isxdigit('F') ? "F is a " : "F is not a ", "hexadecimal digit",
24        isxdigit('J') ? "J is a " : "J is not a ", "hexadecimal digit",
25        isxdigit('7') ? "7 is a " : "7 is not a ", "hexadecimal digit",
26        isxdigit('$') ? "$ is a " : "$ is not a ", "hexadecimal digit",
27        isxdigit('f') ? "f is a " : "f is not a ", "hexadecimal digit");
28 }
```

```

According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit

```

Figura 8.1 Uso delle funzioni `isdigit`, `isalpha`, `isalnum` e `isxdigit`.

Il programma della Figura 8.1 usa l'operatore condizionale (`? :`) per determinare se per ogni carattere esaminato si deve stampare la stringa "is a" o la stringa "is not a". Per esempio, l'espressione

```
isdigit('8') ? "8 is a" : "8 is not a"
```

indica che se '8' è una cifra, viene stampata la stringa "8 is a", e se '8' non è una cifra (cioè, `isdigit` restituisce 0), viene stampata la stringa "8 is not a".

8.3.2 Funzioni `islower`, `isupper`, `tolower` e `toupper`

Il programma della Figura 8.2 illustra le funzioni `islower`, `isupper`, `tolower` e `toupper`. La funzione `islower` determina se il suo argomento è una lettera minuscola (a-z). La funzione `isupper` determina se il suo argomento è una lettera maiuscola (A-Z). La funzione `tolower` converte una lettera maiuscola in una lettera minuscola e restituisce la lettera minuscola. Se l'argomento non è una lettera maiuscola, `tolower` restituisce l'argomento invariato. La funzione `toupper` converte una lettera minuscola in una lettera maiuscola e restituisce la lettera maiuscola. Se l'argomento non è una lettera minuscola, `toupper` restituisce l'argomento invariato.

```

1 // fig08_02.c
2 // Uso delle funzioni islower, isupper, tolower e toupper
3 #include <ctype.h>
4 #include <stdio.h>
5
6 int main(void) {
7     printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to islower:",
8         islower('p') ? "p is a" : "p is not a", "lowercase letter",
9         islower('P') ? "P is a" : "P is not a", "lowercase letter",
10        islower('5') ? "5 is a" : "5 is not a", "lowercase letter",
11        islower('!') ? "!" is a" : "!" is not a", "lowercase letter");
12

```

```

13     printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to isupper:",
14         isupper('D') ? "D is an " : "D is not an ", "uppercase letter",
15         isupper('d') ? "d is an " : "d is not an ", "uppercase letter",
16         isupper('8') ? "8 is an " : "8 is not an ", "uppercase letter",
17         isupper('$') ? "$ is an " : "$ is not an ", "uppercase letter");
18
19     printf("%s%c\n%s%c\n%s%c\n%s%c\n", 
20         "u converted to uppercase is ", toupper('u'),
21         "7 converted to uppercase is ", toupper('7'),
22         "$ converted to uppercase is ", toupper('$'),
23         "L converted to lowercase is ", tolower('L'));
24 }

```

```

According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter

According to isupper:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter

u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l

```

Figura 8.2 Uso delle funzioni `islower`, `isupper`, `tolower` e `toupper`.

8.3.3 Funzioni `isspace`, `iscntrl`, `ispunct`, `isprint` e `isgraph`

Il programma della Figura 8.3 illustra le funzioni `isspace`, `iscntrl`, `ispunct`, `isprint` e `isgraph`. La funzione `isspace` determina se un carattere è uno dei seguenti caratteri di spaziatura: spazio (' '), avanzamento pagina ('\f'), newline ('\n'), ritorno a capo ('\r'), tab orizzontale ('\t') o tab verticale ('\v'). La funzione `iscntrl` determina se un carattere è uno dei seguenti **caratteri di controllo**: tab orizzontale ('\t'), tab verticale ('\v'), avanzamento pagina ('\f'), avviso ('\a'), backspace ('\b'), ritorno a capo ('\r') o newline ('\n'). La funzione `ispunct` determina se un carattere è un **carattere stampabile** diverso da uno spazio, una cifra o una lettera, come \$, #, (,), [], {}, :, : o %. La funzione `isprint` determina se un carattere può essere stampato sullo schermo (incluso il carattere di spazio). La funzione `isgraph` è come `isprint`, però non include il carattere di spazio.

```

1 // fig08_03.c
2 // Uso delle funzioni isspace, iscntrl, ispunct, isprint e isgraph
3 #include <ctype.h>
4 #include <stdio.h>
5
6 int main(void) {
7     printf("%s\n%s%s%s\n%s%s%s\n%s%s\n\n", "According to isspace:",
8         "Newline", isspace('\n') ? " is a " : " is not a ",
9         "whitespace character",
10        "Horizontal tab", isspace('\t') ? " is a " : " is not a ",

```

```

11     "whitespace character",
12     isspace('%') ? "% is a " : "% is not a ", "whitespace character");
13
14 printf("%s\n%s%s\n%s\n", "According to iscntrl:",
15     "\nnewline", iscntrl('\n') ? " is a " : " is not a ",
16     "control character",
17     iscntrl('$') ? "$ is a " : "$ is not a ", "control character");
18
19 printf("%s\n%s%s\n%s\n", "According to ispunct:",
20     ispunct(';') ? ";" is a " : ";" is not a ", "punctuation character",
21     ispunct('Y') ? "Y is a " : "Y is not a ", "punctuation character",
22     ispunct('#') ? "# is a " : "# is not a ", "punctuation character");
23
24 printf("%s\n%s%s\n%s\n", "According to isprint:",
25     isprint('$') ? "$ is a " : "$ is not a ", "printing character",
26     "Alert", isprint('\a') ? "\a is a " : "\a is not a ",
27     "printing character");
28
29 printf("%s\n%s%s\n%s\n", "According to isgraph:",
30     isgraph('Q') ? "Q is a " : "Q is not a ",
31     "printing character other than a space",
32     "Space", isgraph(' ') ? " " is a " : " " is not a ",
33     "printing character other than a space");
34 }

```

```

According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character

According to iscntrl:
\nnewline is a control character
$ is not a control character

According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character

According to isprint:
$ is a printing character
Alert is not a printing character

According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space

```

Figura 8.3 Uso delle funzioni isspace, iscntrl, ispunct, isprint e isgraph.

✓ Autovalutazione

- I. (Scelta multipla)** A quale funzione corrisponde la seguente descrizione: “Restituisce un valore vero se il suo argomento char è una cifra o una lettera; altrimenti, restituisce 0 (falso)”?

- a) isalnum.
- b) isdigit.
- c) isalpha.
- d) isxdigit.

Risposta: a.

2. (*Codice*) Che cosa stampa la seguente istruzione printf?

```
printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalpha:",
       isalpha('X') ? "X is a " : "X is not a ", "letter",
       isalpha('m') ? "m is a " : "m is not a ", "letter",
       isalpha('$') ? "$ is a " : "$ is not a ", "letter",
       isalpha('7') ? "7 is a " : "7 is not a ", "letter");
```

Risposta:

```
According to isalpha:
X is a letter
m is a letter
$ is not a letter
7 is not a letter
```

8.4 Funzioni di conversione di stringhe

Questo paragrafo presenta le **funzioni di conversione di stringhe** della libreria di utilità generale (`<stdlib.h>`). Queste funzioni convertono stringhe di cifre in valori interi e in virgola mobile. La tabella seguente riepiloga le funzioni di conversione di stringhe. Il C standard comprende anche `strtoll` e `strtoull` per convertire stringhe, rispettivamente, in valori `long long int` e `unsigned long long int`.

Prototipo della funzione	Descrizione della funzione
<code>double strtod(const char *nPtr, char **endPtr);</code>	Converte la stringa <code>nPtr</code> in un <code>double</code> .
<code>long strtol(const char *nPtr, char **endPtr, int base);</code>	Converte la stringa <code>nPtr</code> in un <code>long</code> .
<code>unsigned long strtoul(const char *nPtr, char **endPtr, int base);</code>	Converte la stringa <code>nPtr</code> in un <code>unsigned long</code> .

8.4.1 Funzione `strtod`

La funzione `strtod` (Figura 8.4) converte una sequenza di caratteri che rappresentano un valore in virgola mobile in `double`. La funzione restituisce 0 se non è in grado di convertire una porzione del suo primo argomento in `double`. La funzione riceve due argomenti: una stringa (`char *`) e un puntatore a una stringa (`char **`). L'argomento stringa contiene la sequenza di caratteri da convertire in un `double`. I caratteri di spaziatura all'inizio della stringa vengono ignorati. La funzione usa l'argomento `char **` per far sì che un oggetto `char *` nella funzione chiamante (`stringPtr`) punti al primo carattere dopo la porzione convertita della stringa. Se nessuna parte può essere convertita, la funzione indirizza il puntatore all'inizio della stringa. La riga 10 assegna a `d` il valore `double` ottenuto dalla conversione di `string` e indirizza `stringPtr` all'operatore `%` in `string`.

```
1 // fig08_04.c
2 // Uso della funzione strtod
3 #include <stdio.h>
```

```

4 #include <stdlib.h>
5
6 int main(void) {
7     const char *string = "51.2% are admitted";
8     char *stringPtr = NULL;
9
10    double d = strtod(string, &stringPtr);
11
12    printf("The string \"%s\" is converted to the\n", string);
13    printf("double value %.2f and the string \"%s\"\n", d, stringPtr);
14 }

```

The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"

Figura 8.4 Uso della funzione `strtod`.

8.4.2 Funzione `strtol`

La funzione `strtol` (Figura 8.5) converte in un valore `long int` una sequenza di caratteri che rappresenta un intero. La funzione restituisce 0 se non è in grado di convertire una porzione del suo primo argomento in un `long int`. I tre argomenti della funzione sono una stringa (`char *`), un puntatore a una stringa e un intero. Questa funzione opera in modo identico a `strtod`, ma il terzo argomento specifica la *base* del valore da convertire.

```

1 // fig08_05.c
2 // Uso della funzione strtol
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7     const char *string = "-1234567abc";
8     char *remainderPtr = NULL;
9
10    long x = strtol(string, &remainderPtr, 0);
11
12    printf("%s \"%s\"\n%s%ld\n%s \"%s\"\n%s%ld\n".
13           "The original string is ", string,
14           "The converted value is ", x,
15           "The remainder of the original string is ", remainderPtr,
16           "The converted value plus 567 is ", x + 567);
17 }

```

The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000

Figura 8.5 Uso della funzione `strtol`.

La riga 10 assegna a `x` il valore `long` ottenuto dalla conversione di `string` e punta `remainderPtr` alla "a" in `string`. L'uso di `NULL` per il secondo argomento fa sì che il *resto della stringa sia ignorato*. Il terzo argomento, 0, indica che il valore da convertire può essere nel formato ottale (base 8), decimale (base 10) o esadecimale

(base 16). La base può essere specificata come 0 oppure come qualsiasi valore tra 2 e 36.¹ Le rappresentazioni numeriche di interi dalla base 11 alla base 36 usano i caratteri A-Z per rappresentare i valori da 10 a 35. Per esempio, i valori esadecimali possono essere costituiti dalle cifre da 0 a 9 e dai caratteri A-F.

8.4.3 Funzione strtoul

La funzione `strtoul` (Figura 8.6) converte in un valore `unsigned long int` una sequenza di caratteri che rappresenta un valore `unsigned long int`. La funzione opera in maniera identica alla funzione `strtol`. La riga 10 assegna a `x` il valore `unsigned long int` convertito da `string` e indirizza `remainderPtr` alla "a" in `string`. Il terzo argomento, 0, indica che il valore da convertire può essere in formato ottale, decimale o esadecimale.

```

1 // fig08_06.c
2 // Uso della funzione strtoul
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7     const char *string = "1234567abc";
8     char *remainderPtr = NULL;
9
10    unsigned long int x = strtoul(string, &remainderPtr, 0);
11
12    printf("%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
13          "The original string is ", string,
14          "The converted value is ", x,
15          "The remainder of the original string is ", remainderPtr,
16          "The converted value minus 567 is ", x - 567);
17 }
```

```

The original string is "1234567abc"
The converted value is 1234567
The remainder of the original string is "abc"
The converted value minus 567 is 1234000

```

Figura 8.6 Uso della funzione `strtoul`.

✓ Autovalutazione

1. (*Discussione*) Perché la lista dei parametri di una funzione conterrebbe un parametro `char **`?

Risposta: Un `char **` è tipicamente un puntatore a un puntatore `char *` nella funzione chiamante. Una funzione chiamata usa un puntatore di questo tipo per ricevere un `char *` per riferimento e modificarlo nella funzione chiamante (per esempio, per puntarla a un'altra stringa). Questo è un esempio di un puntatore a un puntatore.

2. (*Scelta multipla*) Quale delle seguenti affermazioni relative alla funzione `strtol` è falsa?

- La funzione `strtol` converte in un valore `long int` una sequenza di caratteri che rappresenta un intero, oppure restituisce 0 se non è in grado di convertire in un `long int` alcuna porzione del suo primo argomento.
- I tre argomenti della funzione `strtol` sono una stringa (`char *`), un puntatore a una stringa (`char **`) e un intero.

1. Consultate l'Appendice E (disponibile sulla piattaforma MyLab) per una descrizione dettagliata dei sistemi di numerazione ottale, decimale ed esadecimale.

- c) L'argomento stringa contiene la sequenza di caratteri da convertire in long; i caratteri di spaziatura all'inizio della stringa vengono ignorati.
- d) La funzione usa l'argomento char ** per dare alla funzione chiamante accesso alla porzione numerica della stringa da convertire.

Risposta: d) è falsa. In realtà, la funzione usa l'argomento char ** per modificare un char * nella funzione chiamante in modo che punti alla locazione del primo carattere *dopo* la porzione della stringa che viene convertita. Se nessuna parte può essere convertita, la funzione modifica char * in modo che punti all'intera stringa.

8.5 Funzioni della libreria standard di input/output

Questo paragrafo presenta le funzioni della libreria standard di input/output (<stdio.h>) per la manipolazione di caratteri e stringhe, riepilogate nella tabella seguente.

Prototipo della funzione	Descrizione della funzione
<code>int getchar(void);</code>	Restituisce il carattere successivo dello standard input come un intero.
<code>char *fgets(char *s, int n, FILE *stream);</code>	Legge caratteri dallo stream specificato e li memorizza nell'array s finché non si incontra un carattere newline o end-of-file, oppure finché non vengono letti n - 1 byte. In questo capitolo usiamo lo stream stdin – lo stream standard input – per leggere caratteri dalla tastiera. Un carattere nullo di terminazione è aggiunto in coda all'array. Restituisce la stringa che è stata letta in s. Se si incontra un newline, esso è incluso nella stringa memorizzata.
<code>int putchar(int c);</code>	Stampa il carattere memorizzato in c e lo restituisce come intero.
<code>int puts(const char *s);</code>	Stampa la stringa s seguita da un carattere newline. Restituisce un intero diverso da zero se ha successo o EOF se si verifica un errore.
<code>int sprintf(char *s, const char *format, ...);</code>	Equivalenti a printf, ma l'output è memorizzato nell'array s anziché stampato sullo schermo. Restituisce il numero di caratteri scritti in s se ha successo o EOF se si verifica un errore.
<code>int sscanf(char *s, const char *format, ...);</code>	Equivalenti a scanf, ma l'input è letto dall'array s invece che dalla tastiera. Restituisce il numero di elementi letti con successo dalla funzione o EOF se si verifica un errore.

8.5.1 Funzioni fgets e putchar

Il programma della Figura 8.7 usa funzioni fgets e putchar per leggere una riga di testo dallo standard input (tastiera) e, utilizzando la ricorsione, inviare in uscita i caratteri della riga in ordine inverso. La riga 12 usa la funzione fgets per leggere i caratteri e memorizzarli nel suo array di caratteri ricevuto come argomento finché non incontra un newline o l'indicatore di end-of-file, oppure finché non viene letto il numero massimo di caratteri. Il numero massimo di caratteri è pari al valore specificato nel secondo argomento di fgets meno uno. Il terzo argomento specifica lo stream da cui leggere i caratteri (in questo caso usiamo lo stream standard input, stdin). Quando termina la lettura, fgets aggiunge in coda all'array un carattere nullo ('\0'). La funzione putchar (riga 27) stampa il suo argomento di tipo carattere.

```

1 // fig08_07.c
2 // Uso delle funzioni fgets e putchar
3 #include <stdio.h>
4 #define SIZE 80
5
6 void reverse(const char * const sPtr);
7
8 int main(void) {
9     char sentence[SIZE] = "";
10
11    puts("Enter a line of text:");
12    fgets(sentence, SIZE, stdin); // leggi una riga di testo
13
14    printf("\n%s", "The line printed backward is:");
15    reverse(sentence);
16    puts("");
17 }
18
19 // stampa ricorsivamente i caratteri della stringa in ordine inverso
20 void reverse(const char * const sPtr) {
21     // se si raggiunge la fine della stringa
22     if ('\0' == sPtr[0]) { // caso di base
23         return;
24     }
25     else { // se non e' la fine della stringa
26         reverse(&sPtr[1]); // passo di ricorsione
27         putchar(sPtr[0]); // usa putchar per stampare il carattere
28     }
29 }

```

```

Enter a line of text:
Characters and Strings

The line printed backward is:
sgnirts dna sretcarahC

```

Figura 8.7 Uso delle funzioni fgets e putchar.

Funzione reverse

Il programma chiama la funzione ricorsiva `reverse`² per stampare la riga di testo all'indietro. Se il primo carattere dell'array è il carattere nullo '\0', `reverse` termina. Altrimenti, `reverse` chiama ricorsivamente se stessa con l'indirizzo del sottoarray che inizia all'elemento `sPtr[1]`. La riga 27 invia in uscita il carattere all'elemento `sPtr[0]` quando la chiamata ricorsiva è completata. L'ordine delle due istruzioni nelle righe 26 e 27 fa sì che `reverse` proceda verso il carattere nullo di terminazione della stringa *prima* di stampare un carattere. Quando le chiamate ricorsive sono completate, i caratteri sono inviati in uscita in ordine inverso.

8.5.2 Funzione getchar

Il programma della Figura 8.8 usa funzioni `getchar` per leggere un carattere alla volta dallo standard input nell'array di caratteri `sentence`, poi usa `puts` per stampare i caratteri come stringa. La funzione `getchar`

2. Usiamo la ricorsione in questo contesto per scopi dimostrativi. Solitamente è più efficiente l'uso di un ciclo per l'iterazione dall'ultimo carattere (quello alla posizione corrispondente alla lunghezza della stringa diminuita di 1) al primo carattere (quello alla posizione 0) di una stringa.

legge un carattere dallo standard input e restituisce il carattere come intero. Come potete ricordare dal Paragrafo 4.6, un intero viene restituito per supportare l'indicatore di end-of-file. Come è noto, la funzione `puts` riceve una stringa come argomento e la stampa seguita da un carattere newline. Il programma smette di leggere caratteri quando sono stati letti 79 caratteri o quando `getchar` legge un carattere newline. La riga 18 aggiunge un carattere nullo a `sentence` per terminare la stringa. Poi la riga 21 usa `puts` per stampare `sentence`.

```

1 // fig08_08.c
2 // Uso della funzione getchar
3 #include <stdio.h>
4 #define SIZE 80
5
6 int main(void) {
7     int c = 0; // variabile che contiene il carattere inserito dall'utente
8     char sentence[SIZE] = "";
9     int i = 0;
10
11    puts("Enter a line of text:");
12
13    // usa getchar per leggere ogni carattere
14    while ((i < SIZE - 1) && (c = getchar()) != '\n') {
15        sentence[i++] = c;
16    }
17
18    sentence[i] = '\0'; // termina la stringa
19
20    puts("\nThe line entered was:");
21    puts(sentence); // stampa la stringa
22 }
```

Enter a line of text:
This is a test.

The line entered was:
This is a test.

Figura 8.8 Uso della funzione `getchar`.

8.5.3 Funzione `sprintf`

Il programma della Figura 8.9 usa la funzione `sprintf` per memorizzare dati formattati nell'array di caratteri `s`. La funzione usa le stesse specifiche di conversione di `printf` (vedi Capitolo 9 per un'analisi dettagliata della formattazione). Il programma legge un valore `int` e un valore `double` da formattare e memorizzare nell'array `s`. L'array `s` è il primo argomento di `sprintf`.

```

1 // fig08_09.c
2 // Uso della funzione sprintf
3 #include <stdio.h>
4 #define SIZE 80
5
6 int main(void) {
7     int x = 0;
8     double y = 0.0;
9
10    puts("Enter an integer and a double:");

```

```

11     scanf("%d%lf", &x, &y);
12
13     char s[SIZE] = {'\0'}; // crea un array di char
14     sprintf(s, "integer:%6d\ndouble:%7.2f", x, y);
15
16     printf("The formatted output stored in array s is:\n%s\n", s);
17 }

```

```

Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer: 298
double: 87.38

```

Figura 8.9 Uso della funzione `sprintf`.

8.5.4 Funzione `sscanf`

Il programma della Figura 8.10 illustra la funzione `sscanf`, che opera come la funzione `scanf` ma legge dati formattati da una stringa. Il programma legge un `int` e un `double` dall'array di caratteri `s`, li memorizza in `x` e `y`, poi li stampa.

```

1 // fig08_10.c
2 // Uso della funzione sscanf
3 #include <stdio.h>
4
5 int main(void) {
6     char s[] = "31298 87.375";
7     int x = 0;
8     double y = 0;
9
10    sscanf(s, "%d%lf", &x, &y);
11    puts("The values stored in character array s are:");
12    printf("integer:%6d\ndouble:%8.3f\n", x, y);
13 }

```

```

The values stored in character array s are:
integer: 31298
double: 87.375

```

Figura 8.10 Uso della funzione `sscanf`.

✓ Autovalutazione

1. (*Scelta multipla*) Quale funzione ha la descrizione seguente: “Stampa il carattere memorizzato nel suo parametro e lo restituisce come intero”?

- a) `getchar`.
- b) `sprintf`.
- c) `puts`.
- d) `putchar`.

Risposta: d.

2. (*Vero/Falso*) La funzione `getchar` legge un carattere dallo standard input e lo restituisce come tipo `char`.

Risposta: Falso. In realtà, `getchar` restituisce un intero per supportare l'indicatore di end-of-file, che è -1.

8.6 Funzioni per la manipolazione di stringhe della libreria per il trattamento delle stringhe

La libreria per il trattamento delle stringhe (`<string.h>`) fornisce molte funzioni utili per:

- manipolare stringhe (**copiare stringhe e concatenarle**);
- **confrontare stringhe**;
- ricercare all'interno di stringhe caratteri e altre stringhe;
- **suddividere le stringhe in token** (separare le stringhe in pezzi logici);
- determinare la **lunghezza delle stringhe**.

Questo paragrafo presenta le funzioni per la manipolazione di stringhe della libreria per il trattamento delle stringhe, che sono riepilogate nella tabella seguente. Eccetto `strncpy`, ogni funzione aggiunge il carattere nullo in coda al suo risultato.

Prototipo della funzione	Descrizione della funzione
<code>char *strcpy(char *s1, const char *s2)</code>	Copia la stringa <code>s2</code> nell'array <code>s1</code> e restituisce <code>s1</code> .
<code>char *strncpy(char *s1, const char *s2, size_t n)</code>	Copia al massimo <code>n</code> caratteri della stringa <code>s2</code> nell'array <code>s1</code> e restituisce <code>s1</code> .
<code>char *strcat(char *s1, const char *s2)</code>	Aggiunge la stringa <code>s2</code> in coda all'array <code>s1</code> e restituisce <code>s1</code> . Il primo carattere della stringa <code>s2</code> sovrascrive il carattere nullo di terminazione di <code>s1</code> .
<code>char *strncat(char *s1, const char *s2, size_t n)</code>	Aggiunge al massimo <code>n</code> caratteri della stringa <code>s2</code> in coda all'array <code>s1</code> e restituisce <code>s1</code> . Il primo carattere della stringa <code>s2</code> sovrascrive il carattere nullo di terminazione di <code>s1</code> .

Le funzioni `strncpy` e `strncat` specificano un parametro di tipo `size_t`. La funzione `strcpy` copia la stringa del secondo argomento nell'array `char` nel suo primo argomento. Dovete assicurarsi che l'array sia grande abbastanza da memorizzare la stringa e il suo carattere nullo di terminazione, che viene anche copiato. La funzione `strncpy` è equivalente a `strcpy` ma copia solo il numero di caratteri specificato. *La funzione strncpy non copia il carattere nullo di terminazione del suo secondo argomento a meno che il numero di caratteri da copiare sia maggiore della lunghezza della stringa.* Per esempio, se "test" è il secondo argomento, un carattere nullo di terminazione viene scritto solo se il terzo argomento di `strncpy` è almeno 5 (quattro caratteri in "test" più un carattere nullo di terminazione). Se il terzo argomento è più grande di 5, alcune implementazioni aggiungono in coda all'array tutti i caratteri nulli necessari per scrivere il numero totale di caratteri specificato dal terzo argomento. Altre implementazioni si arrestano dopo la scrittura del primo carattere nullo. È un errore logico non aggiungere in coda un carattere nullo di terminazione al primo argomento di una `strncpy` quando il terzo argomento è minore o uguale alla lunghezza della stringa del secondo argomento.



8.6.1 Funzioni `strcpy` e `strncpy`

Il programma della Figura 8.11 usa `strcpy` per copiare l'intera stringa dell'array `x` nell'array `y` e `strncpy` per copiare i primi 14 caratteri dell'array `x` nell'array `z`. La riga 19 aggiunge un carattere nullo ('\0') in coda all'array `z` poiché la chiamata di `strncpy` non scrive un carattere nullo di terminazione – il terzo argomento è minore della lunghezza della stringa del secondo argomento.

```

1 // fig08_11.c
2 // Uso delle funzioni strcpy e strncpy
3 #include <stdio.h>
4 #include <string.h>
5 #define SIZE1 25
6 #define SIZE2 15
7
8 int main(void) {
9     char x[] = "Happy Birthday to You"; // inizializza l'array di caratteri x
10    char y[SIZE1] = ""; // crea un array di caratteri y
11    char z[SIZE2] = ""; // crea un array di caratteri z
12
13    // copia il contenuto di x in y
14    printf("%s\n%s\n",
15           "The string in array x is: ", x,
16           "The string in array y is: ", strcpy(y, x));
17
18    strncpy(z, x, SIZE2 - 1); // copia i primi 14 caratteri di x in z
19    z[SIZE2 - 1] = '\0'; // termina la stringa in z, perche' '\0' non copiato
20    printf("The string in array z is: %s\n", z);
21 }

```

```

The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday

```

Figura 8.11 Uso delle funzioni strcpy e strncpy.

8.6.2 Funzioni strcat e strncat

La funzione **strcat** aggiunge la stringa del suo secondo argomento in coda alla stringa del suo primo argomento dell'array di caratteri, sostituendo il carattere nullo ('\0') del primo argomento. *Dovete essere sicuri che l'array usato per memorizzare la prima stringa sia grande abbastanza da memorizzare la prima stringa, la seconda stringa e il carattere nullo di terminazione copiato dalla seconda stringa.* La funzione **strncat** aggiunge un numero specificato di caratteri della seconda stringa in coda alla prima stringa e aggiunge un carattere di terminazione '\0'. Il programma della Figura 8.12 illustra l'uso della funzione **strcat** e della funzione **strncat**.

```

1 // fig08_12.c
2 // Uso delle funzioni strcat e strncat
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     char s1[20] = "Happy "; // inizializza l'array di caratteri s1
8     char s2[] = "New Year "; // inizializza l'array di caratteri s2
9     char s3[40] = ""; // inizializza l'array di caratteri s3 come vuoto
10
11    printf("s1 = %s\ns2 = %s\n", s1, s2);
12
13    // aggiungi s2 in coda a s1
14    printf("strcat(s1, s2) = %s\n", strcat(s1, s2));
15

```

```

16 // aggiungi i primi 6 caratteri di s1 in coda a s3
17 printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));
18
19 // aggiungi s1 in coda a s3
20 printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
21 }

```

```

s1 = Happy
s2 = New Year
strcat(s1, s2) = Happy New Year
strncat(s3, s1, 6) = Happy
strcat(s3, s1) = Happy Happy New Year

```

Figura 8.12 Uso delle funzioni `strcat` e `strncat`.

✓ Autovalutazione

- (Scelta multipla) Quale delle seguenti affermazioni relative alle funzioni `strcat` e `strncat` è falsa?
 - La funzione `strcat` aggiunge la stringa ricevuta come suo secondo argomento in coda alla stringa nell'array di caratteri ricevuto come suo primo argomento.
 - Il primo carattere del secondo argomento di `strcat` è posto immediatamente dopo il carattere nullo ('\0') che termina la stringa nel primo argomento.
 - Dovete assicurarvi che l'array che contiene la prima stringa sia grande abbastanza da memorizzare la prima stringa, la seconda stringa e il carattere di terminazione '\0' copiato dalla seconda stringa.
 - La funzione `strncat` aggiunge un numero specificato di caratteri della seconda stringa in coda alla prima. Un carattere di terminazione '\0' è aggiunto automaticamente in coda al risultato.

Risposta: b) è falsa. In realtà, il primo carattere del secondo argomento di `strcat` sostituisce il carattere nullo ('\0') che termina la stringa nel primo argomento.

- (Completare) La funzione `strcpy` copia il suo secondo argomento (una stringa) nel suo primo argomento, un array di caratteri che deve essere _____.

Risposta: grande abbastanza da memorizzare la stringa, incluso il suo carattere nullo di terminazione.

8.7 Funzioni di confronto della libreria per il trattamento delle stringhe

Questo paragrafo presenta le **funzioni di confronto di stringhe** della libreria per il trattamento delle stringhe `strcmp` e `strncmp`, riepilogate qui di seguito.

Prototipo della funzione	Descrizione della funzione
<code>int strcmp(const char *s1, const char *s2);</code>	<i>Confronta</i> la stringa <code>s1</code> con la stringa <code>s2</code> . La funzione restituisce 0, un valore minore di 0 o maggiore di 0 se <code>s1</code> è, rispettivamente, uguale, minore o maggiore di <code>s2</code> .
<code>int strncmp(const char *s1, const char *s2, size_t n);</code>	<i>Confronta</i> fino a <code>n</code> caratteri della stringa <code>s1</code> con la stringa <code>s2</code> . La funzione restituisce 0, un valore minore di 0 o maggiore di 0 se <code>s1</code> è, rispettivamente, uguale, minore o maggiore di <code>s2</code> .

Il programma della Figura 8.13 confronta tre stringhe usando `strcmp` e `strncmp`. La funzione `strcmp` confronta il suo primo argomento con il secondo argomento, carattere per carattere. La funzione restituisce:

- 0 se le stringhe sono uguali;
- un *valore negativo* se la prima stringa è minore della seconda;
- un *valore positivo* se la prima stringa è maggiore della seconda.

La funzione `strncmp` è equivalente a `strcmp` ma confronta fino a un numero specificato di caratteri. La funzione `strncmp` *non* confronta i caratteri che seguono un carattere nullo in una stringa. Il programma stampa il valore intero restituito da ogni chiamata delle funzioni.

```

1 // fig08_13.c
2 // Uso delle funzioni strcmp e strncmp
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *s1 = "Happy New Year"; // inizializza un puntatore a char
8     const char *s2 = "Happy New Year"; // inizializza un puntatore a char
9     const char *s3 = "Happy Holidays"; // inizializza un puntatore a char
10
11    printf("s1 = %s\ns2 = %s\ns3 = %s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12          s1, s2, s3,
13          "strcmp(s1, s2) = ", strcmp(s1, s2),
14          "strcmp(s1, s3) = ", strcmp(s1, s3),
15          "strcmp(s3, s1) = ", strcmp(s3, s1));
16
17    printf("%s%2d\n%s%2d\n%s%2d\n",
18          "strncmp(s1, s3, 6) = ", strncmp(s1, s3, 6),
19          "strncmp(s1, s3, 7) = ", strncmp(s1, s3, 7),
20          "strncmp(s3, s1, 7) = ", strncmp(s3, s1, 7));
21 }
```

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays
```

```
strcmp(s1, s2) = 0
strcmp(s1, s3) = 1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) = 0
strncmp(s1, s3, 7) = 1
strncmp(s3, s1, 7) = -1
```

Figura 8.13 Uso delle funzioni `strcmp` e `strncmp`.

Confrontare le stringhe

Per capire cosa significa per una stringa essere “maggiori di” o “minori di” un’altra, considerate la procedura che mette in ordine alfabetico una serie di cognomi. Voi, indubbiamente, mettereste “Jones” prima di “Smith” perché la prima lettera di “Jones” viene prima della prima lettera di “Smith” nell’alfabeto. Ma l’alfabeto è più che una semplice lista di 26 lettere: è una lista ordinata di caratteri. Ogni lettera compare nella lista in

una posizione specifica. La "Z" è più che una semplice lettera dell'alfabeto; la "Z" è specificamente la 26ma lettera dell'alfabeto. Ricordate inoltre che le lettere minuscole hanno un valore numerico più alto delle lettere maiuscole, pertanto "a" è maggiore di "A".

Come sanno le funzioni di confronto di stringhe che una particolare lettera viene prima di un'altra? Tutti i caratteri sono rappresentati nel computer come **codici numerici** in un insieme di caratteri come ASCII e Unicode; quando il computer confronta due stringhe, confronta in realtà i codici numerici dei caratteri in ciascuna stringa. Questo si chiama confronto lessicografico. Consultate l'Appendice B per i valori numerici dei caratteri ASCII. ASCII è un sottoinsieme dell'insieme di caratteri Unicode.

I valori negativi e positivi restituiti dalle funzioni `strcmp` e `strncmp` sono *dipendenti dall'implementazione*. Per alcune implementazioni, questi valori sono -1 o 1, come nella Figura 8.13. Per altre, i valori restituiti sono la differenza tra i codici numerici dei primi caratteri diversi in ogni stringa. Per i confronti in questo programma, si tratta della differenza tra i codici numerici di "N" in "New" e "H" in "Holidays" (6 o -6, a seconda di quale stringa è il primo argomento).

✓ Autovalutazione

- (Scelta multipla)** Quale delle seguenti affermazioni relative alle funzioni `strcmp` e `strncmp` è falsa?
 - La funzione `strcmp` confronta la stringa che riceve come primo argomento con la stringa che riceve come secondo argomento, carattere per carattere.
 - La funzione `strcmp` restituisce 0 se le stringhe sono uguali, un valore negativo se la prima stringa è minore della seconda e un valore positivo se la prima stringa è maggiore della seconda.
 - La funzione `strncmp` è equivalente a `strcmp` ma confronta fino a un numero specificato di caratteri.
 - La funzione `strncmp` confronta i caratteri che seguono un carattere nullo in una stringa.

Risposta: d) è falsa. In realtà, la funzione `strncmp` non confronta i caratteri che seguono un carattere nullo in una stringa.

- (Discussione)** Come sanno le funzioni di confronto di stringhe `strcmp` e `strncmp` che una particolare lettera "viene prima" di un'altra?

Risposta: Tutti i caratteri sono rappresentati nel computer come codici numerici in un insieme di caratteri come ASCII e Unicode; quando il computer confronta due stringhe, confronta in realtà i codici numerici dei caratteri. Questo si chiama confronto lessicografico.

8.8 Funzioni per la ricerca della libreria per il trattamento delle stringhe

Questo paragrafo presenta le funzioni della libreria per il trattamento delle stringhe usate per effettuare ricerche all'interno di stringhe per individuare caratteri e altre stringhe, riepilogate nella tabella seguente.

Prototipi e descrizioni delle funzioni
<code>char *strchr(const char *s, int c);</code> Localizza la prima occorrenza del carattere c nella stringa s. Se c viene trovato, <code>strchr</code> restituisce un puntatore a c in s. Altrimenti, viene restituito un puntatore NULL.
<code>size_t strcspn(const char *s1, const char *s2);</code> Determina e restituisce la lunghezza del segmento iniziale della stringa s1 costituito da caratteri non contenuti nella stringa s2.
<code>size_t strspn(const char *s1, const char *s2);</code> Determina e restituisce la lunghezza del segmento iniziale della stringa s1 costituito solo da caratteri contenuti nella stringa s2.

➤ **char *strpbrk(const char *s1, const char *s2);**
 Localizza la prima occorrenza di un carattere della stringa s2 nella stringa s1. Se viene trovato un carattere della stringa s2, strpbrk restituisce un puntatore a quel carattere in s1. Altrimenti, restituisce NULL.

char *strrchr(const char *s, int c);
 Localizza l'ultima occorrenza di c nella stringa s. Se c viene trovato, strrchr restituisce un puntatore a c nella stringa s. Altrimenti, restituisce NULL.

char *strstr(const char *s1, const char *s2);
 Localizza la prima occorrenza nella stringa s1 della stringa s2. Se la stringa viene trovata, strstr restituisce un puntatore alla stringa in s1. Altrimenti, restituisce NULL.

char *strtok(char *s1, const char *s2);
 Una sequenza di chiamate a strtok suddivide la stringa s1 in token separati da caratteri contenuti nella stringa s2. I token sono unità logiche, come le parole in una riga di testo. La prima chiamata utilizza s1 come primo argomento, mentre le successive chiamate per continuare a suddividere in token la stessa stringa richiedono NULL come primo argomento. A ogni chiamata viene restituito un puntatore al token corrente. Se non vi sono più token, strtok restituisce NULL.

8.8.1 Funzione strchr

La funzione strchr cerca la prima occorrenza di un carattere in una stringa. Se il carattere viene trovato, strchr restituisce un puntatore al carattere nella stringa; altrimenti, strchr restituisce NULL. Il programma della Figura 8.14 cerca le prime occorrenze di 'a' e 'z' in "This is a test".

```

1 // fig08_14.c
2 // Uso della funzione strchr
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *string = "This is a test"; // inizializza un puntatore a char
8     char character1 = 'a';
9     char character2 = 'z';
10
11    // se character1 e' stato trovato in string
12    if (strchr(string, character1) != NULL) { // puoi rimuovere "!= NULL"
13        printf("\'%c\' was found in \"%s\".\n", character1, string);
14    }
15    else { // se character1 non e' stato trovato
16        printf("\'%c\' was not found in \"%s\".\n", character1, string);
17    }
18
19    // se character2 e' stato trovato in string
20    if (strchr(string, character2) != NULL) { // puoi rimuovere "!= NULL"
21        printf("\'%c\' was found in \"%s\".\n", character2, string);
22    }
23    else { // se character2 non e' stato trovato
24        printf("\'%c\' was not found in \"%s\".\n", character2, string);
25    }
26 }
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```

Figura 8.14 Uso della funzione strchr.

8.8.2 Funzione strcspn

La funzione `strcspn` (Figura 8.15) determina la lunghezza della parte iniziale della stringa nel suo primo argomento che *non* contiene alcun carattere appartenente alla stringa nel suo secondo argomento. La funzione restituisce la lunghezza del segmento.

```
1 // fig08_15.c
2 // Uso della funzione strcspn
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     // inizializza due puntatori a char
8     const char *string1 = "The value is 3.14159";
9     const char *string2 = "1234567890";
10
11    printf("string1 = %s\nstring2 = %s\n\n%s\n%s%zu\n",
12          string1, string2,
13          "The length of the initial segment of string1",
14          "containing no characters from string2 = ",
15          strcspn(string1, string2));
16 }
```

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

Figura 8.15 Uso della funzione strcspn.

8.8.3 Funzione strpbrk

La funzione `strpbrk` cerca nel suo primo argomento, una stringa, la *prima occorrenza* di un qualsiasi carattere che fa parte del suo secondo argomento, un'altra stringa. Se viene trovato un carattere che fa parte del secondo argomento, `strpbrk` restituisce un puntatore al carattere nel primo argomento, altrimenti restituisce `NULL`. Il programma della Figura 8.16 localizza in `string1` la prima occorrenza di un carattere di `string2`.

```
1 // fig08_16.c
2 // Uso della funzione strpbrk
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *string1 = "This is a test";
8     const char *string2 = "beware";
9
10    printf("%s\"%s\"\n%c'%s \"%s\"\n",
11          string2, *strpbrk(string1, string2),
```

```

12     " appears earliest in ", string1);
13 }

```

Of the characters in "beware"
'a' appears earliest in "This is a test"

Figura 8.16 Uso della funzione `strpbrk`.

8.8.4 Funzione `strrchr`

La funzione `strrchr` cerca in una stringa l'ultima occorrenza del carattere specificato. Se il carattere viene trovato, `strrchr` restituisce un puntatore al carattere nella stringa; altrimenti, restituisce `NULL`. La Figura 8.17 mostra un programma che localizza l'ultima occorrenza del carattere 'z' nella stringa "A zoo has many animals including zebras".

```

1 // fig08_17.c
2 // Uso della funzione strrchr
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *s1 = "A zoo has many animals including zebras";
8     int c = 'z'; // carattere da cercare
9
10    printf("%s '%c' %s\n \"%s\"\n",
11           "Remainder of s1 beginning with the last occurrence of character",
12           c, strrchr(s1, c));
13 }

```

Remainder of s1 beginning with the last occurrence of character 'z' is:
"zebras"

Figura 8.17 Uso della funzione `strrchr`.

8.8.5 Funzione `strspn`

La funzione `strspn` (Figura 8.18) determina la lunghezza della parte iniziale della stringa nel suo primo argomento che contiene solo caratteri della stringa nel suo secondo argomento. La funzione restituisce la lunghezza del segmento.

```

1 // fig08_18.c
2 // Uso della funzione strspn
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *string1 = "The value is 3.14159";
8     const char *string2 = "aehi lsTuv";
9
10    printf("string1 = %s\nstring2 = %s\n\n%s\n%zu\n", string1, string2,
11           "The length of the initial segment of string1",
12           "containing only characters from string2 = ",
13           strspn(string1, string2));
14 }

```

```
string1 = The value is 3.14159
string2 = aehi lsTuv
The length of the initial segment of string1
containing only characters from string2 = 13
```

Figura 8.18 Uso della funzione strspn.

8.8.6 Funzione strstr

La funzione **strstr** cerca la prima occorrenza della stringa che riceve come secondo argomento nella stringa che riceve come primo argomento. Se la seconda stringa viene trovata all'interno della prima, **strstr** restituisce un puntatore alla posizione della seconda stringa nel primo argomento. Il programma della Figura 8.19 usa **strstr** per trovare la stringa "def" nella stringa "abcdefabcdef".

```
1 // fig08_19.c
2 // Uso della funzione strstr
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *string1 = "abcdefabcdef";
8     const char *string2 = "def"; // stringa da cercare
9
10    printf("string1 = %s\nstring2 = %s\n\n%s\n%s\n", string1, string2,
11          "The remainder of string1 beginning with the",
12          "first occurrence of string2 is: ", strstr(string1, string2));
13 }
```

```
string1 = abcdefabcdef
string2 = def
The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

Figura 8.19 Uso della funzione strstr.

8.8.7 Funzione strtok

La funzione **strtok** (Figura 8.20) viene usata per suddividere una stringa in una serie di **token**. Un token è una sequenza di caratteri separati da **delimitatori**, come *spazi* o *segni di interruzione*. Qualunque carattere può essere un delimitatore. Per esempio, in una riga di testo, ogni parola può essere considerata un token, e gli spazi e la punteggiatura che separano le parole possono essere considerati delimitatori. Potete modificare la stringa di delimitatori in ogni chiamata a **strtok**. Il programma della Figura 8.20 suddivide in token la stringa "This is a sentence with 7 tokens" e stampa i token. La funzione **strtok** modifica la stringa in ingresso inserendo '\0' alla fine di ogni token, quindi copiate la stringa se avete intenzione di usarla dopo le chiamate a **strtok**. Consultate la raccomandazione del CERT STR06-C che analizza i problemi ipotizzando che **strtok** non modifichi la stringa nel suo primo argomento.

```
1 // fig08_20.c
2 // Uso della funzione strtok
3 #include <stdio.h>
4 #include <string.h>
```

```

5
6 int main(void) {
7     char string[] = "This is a sentence with 7 tokens";
8
9     printf("The string to be tokenized is:\n%s\n\n", string);
10    puts("The tokens are:");
11
12    char *tokenPtr = strtok(string, " "); // trova il primo token
13
14    // continua a suddividere la stringa in token finché tokenPtr non diventa NULL
15    while (tokenPtr != NULL) {
16        printf("%s\n", tokenPtr);
17        tokenPtr = strtok(NULL, " "); // trova il successivo token
18    }
19 }
```

```

The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

Figura 8.20 Uso della funzione `strtok`.

Prima chiamata a `strtok`

Sono necessarie più chiamate a `strtok` per suddividere in token una stringa, supponendo che essa contenga più di un token. La prima chiamata a `strtok` (riga 12) riceve come argomenti una stringa da suddividere in token e una stringa contenente caratteri che separano i token. L'istruzione

```
char *tokenPtr = strtok(string, " "); // trova il primo token
```

assegna a `tokenPtr` un puntatore al primo token in `string`. Il secondo argomento, " ", indica che i token sono separati da spazi. La funzione `strtok` cerca il primo carattere in `string` che non è un carattere di delimitazione (spazio). Questo inizia il primo token. La funzione poi trova il successivo carattere di delimitazione nella stringa e lo sostituisce con un carattere nullo ('\0') per terminare il token corrente. La funzione `strtok` salva un puntatore al carattere successivo a quel token in `string` e restituisce un puntatore al token corrente.

Successive chiamate a `strtok`

Le successive chiamate a `strtok` nella riga 17 continuano a suddividere `string` in token. Queste chiamate ricevono `NULL` come loro primo argomento per indicare che devono continuare la suddivisione in token a partire dalla locazione in `string` salvata nell'ultima chiamata. Se non rimane alcun token, `strtok` restituisce `NULL`.

✓ Autovalutazione

1. (*Scelta multipla*) A quale funzione corrisponde la seguente descrizione: "Localizza la prima occorrenza della stringa `s2` nella stringa `s1`. Se la stringa viene trovata, la funzione restituisce un puntatore alla stringa in `s1`, altrimenti restituisce `NULL`"?

- a) `strpbrk`.
- b) `strstr`.
- c) `strspn`.
- d) `strcspn`.

Risposta: b) `strstr`.

2. (*Completare*) Nel contesto di una funzione `strtok`, un _____ è una sequenza di caratteri separati da delimitatori.

Risposta: token.

8.9 Funzioni di gestione della memoria della libreria per il trattamento delle stringhe

Le funzioni della libreria per il trattamento delle stringhe presentate in questo paragrafo effettuano manipolazioni, confronti e ricerche in blocchi di memoria. Queste funzioni trattano i blocchi di memoria come array di caratteri e possono manipolare qualunque blocco di dati. La tabella seguente riepiloga le funzioni di gestione della memoria della libreria per il trattamento delle stringhe. Nelle analisi delle funzioni, l'“oggetto” si riferisce a un blocco di dati.

Prototipo della funzione	Descrizione della funzione
<code>void *memcpy(void *s1, const void *s2, size_t n);</code>	Copia <code>n</code> byte dall'oggetto puntato da <code>s2</code> nell'oggetto puntato da <code>s1</code> , poi restituisce un puntatore all'oggetto risultante.
<code>void *memmove(void *s1, const void *s2, size_t n);</code>	Copia <code>n</code> byte dall'oggetto puntato da <code>s2</code> nell'oggetto puntato da <code>s1</code> . La copia è eseguita come se i byte fossero dapprima copiati dall'oggetto puntato da <code>s2</code> in un array temporaneo e poi dall'array temporaneo nell'oggetto puntato da <code>s1</code> . Viene restituito un puntatore all'oggetto risultante.
<code>int memcmp(const void *s1, const void *s2, size_t n);</code>	Confronta i primi <code>n</code> byte degli oggetti puntati da <code>s1</code> e <code>s2</code> . La funzione restituisce 0, un numero minore di 0 o maggiore di 0 se <code>s1</code> è, rispettivamente, uguale, minore o maggiore di <code>s2</code> .
<code>void *memchr(const void *s, int c, size_t n);</code>	Localizza la prima occorrenza di <code>c</code> (convertito in un <code>unsigned char</code>) nei primi <code>n</code> byte dell'oggetto puntato da <code>s</code> . Se <code>c</code> viene trovato, <code>memchr</code> restituisce un puntatore a <code>c</code> nell'oggetto; altrimenti, restituisce <code>NULL</code> .
<code>void *memset(void *s, int c, size_t n);</code>	Copia <code>c</code> (convertito in un <code>unsigned char</code>) nei primi <code>n</code> byte dell'oggetto puntato da <code>s</code> , poi restituisce un puntatore al risultato.

I parametri puntatore sono dichiarati `void *`, per cui possono essere usati per manipolare la memoria per qualsiasi tipo di dati. Nel Capitolo 7 abbiamo visto che un puntatore a qualsiasi tipo di dati può essere assegnato direttamente a un puntatore di tipo `void *`, e un puntatore di tipo `void *` può essere assegnato direttamente a un puntatore a qualsiasi tipo di dati. Dal momento che un puntatore a `void *` non può essere dereferenziato, ogni funzione riceve un argomento che specifica il numero di byte che la funzione deve elaborare. Per semplicità, gli esempi di questo paragrafo manipolano array di caratteri (blocchi di caratteri). Le funzioni nella precedente tabella *non* controllano i caratteri nulli di terminazione, poiché manipolano blocchi di memoria che non sono necessariamente stringhe.

8.9.1 Funzione `memcpy`

La funzione `memcpy` copia un numero specificato di byte dall'oggetto puntato dal suo secondo argomento nell'oggetto puntato dal suo primo argomento. La funzione può ricevere un puntatore a qualunque tipo di oggetto. Il suo risultato è *indefinito* se i due oggetti si sovrappongono in memoria, cioè se sono parti dello stesso oggetto. In tali casi, usate invece `memmove`. Il programma della Figura 8.21 usa `memcpy` per copiare la stringa dall'array `s2` nell'array `s1`. La funzione `memcpy` è più efficiente di `strcpy` quando si conosce la dimensione della stringa che si sta copiando.

```

1 // fig08_21.c
2 // Uso della funzione memcpy
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     char s1[17] = "";
8     char s2[] = "Copy this string";
9
10    memcpy(s1, s2, 17); // 17 quindi copiamo il carattere di terminazione \0 di s2
11    puts("After s2 is copied into s1 with memcpy, s1 contains:");
12    puts(s1);
13 }
```

```
After s2 is copied into s1 with memcpy, s1 contains:
Copy this string
```

Figura 8.21 Uso della funzione `memcpy`.

8.9.2 Funzione `memmove`

Come `memcpy`, la funzione `memmove` copia un numero specificato di byte dall'oggetto puntato dal suo secondo argomento nell'oggetto puntato dal suo primo argomento. La copia è eseguita come se i byte fossero copiati dal secondo argomento in un array di caratteri temporaneo, quindi copiati dall'array temporaneo nel primo argomento. Questo permette di copiare i byte da una parte di una stringa (o blocco di memoria) in un'altra parte della stessa stringa (o blocco di memoria), anche se le due porzioni si sovrappongono. Diversamente da `memmove`, le funzioni di manipolazione di stringhe che copiano caratteri producono risultati indefiniti quando copiano tra parti della stessa stringa. Il programma della Figura 8.22 usa `memmove` per copiare gli ultimi 10 byte dell'array `x` nei primi 10 byte dello stesso array.

```

1 // fig08_22.c
2 // Uso della funzione memmove
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     char x[] = "Home Sweet Home"; // inizializza l'array di char x
8
9     printf("The string in array x before memmove is: %s\n", x);
10    printf("The string in array x after memmove is: %s\n",
11          (char *) memmove(x, &x[5], 10));
12 }
```

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is: Sweet Home Home
```

Figura 8.22 Uso della funzione `memmove`.

8.9.3 Funzione memcmp

La funzione **memcmp** (Figura 8.23) confronta il numero specificato di byte del suo primo argomento con i byte corrispondenti del suo secondo argomento. La funzione restituisce un valore maggiore di 0 se il primo argomento è maggiore del secondo, restituisce 0 se gli argomenti sono uguali e restituisce un valore minore di 0 se il primo argomento è minore del secondo.

```

1 // fig08_23.c
2 // Uso della funzione memcmp
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     char s1[] = "ABCDEFG";
8     char s2[] = "ABCDXYZ";
9
10    printf("s1 = %s\ns2 = %s\n\n%s%2d\n%s%2d\n%s%2d\n",
11           s1, s2,
12           "memcmp(s1, s2, 4) = ", memcmp(s1, s2, 4),
13           "memcmp(s1, s2, 7) = ", memcmp(s1, s2, 7),
14           "memcmp(s2, s1, 7) = ", memcmp(s2, s1, 7));
15 }
```

```

s1 = ABCDEFG
s2 = ABCDXYZ

memcmp(s1, s2, 4) = 0
memcmp(s1, s2, 7) = -1
memcmp(s2, s1, 7) = 1
```

Figura 8.23 Uso della funzione **memcmp**.

8.9.4 Funzione memchr

La funzione **memchr** localizza la prima occorrenza di un byte, rappresentato come **unsigned char**, nel numero specificato di byte di un oggetto. Se il byte viene trovato, **memchr** restituisce un puntatore al byte nell'oggetto; altrimenti, restituisce **NULL**. Il programma della Figura 8.24 cerca il byte contenente 'r' nella stringa "This is a string".

```

1 // fig08_24.c
2 // Uso della funzione memchr
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *s = "This is a string";
8
9     printf("The remainder of s after character 'r' is found is \"%s\"\n",
10           (char *) memchr(s, 'r', 16));
11 }
```

```
The remainder of s after character 'r' is found is "ring"
```

Figura 8.24 Uso della funzione **memchr**.

8.9.5 Funzione memset

 La funzione `memset` copia il valore del byte nel suo secondo argomento nei primi *n* byte dell'oggetto puntato dal suo primo argomento, dove *n* è specificato dal terzo argomento. Potete usare `memset` per impostare gli elementi di un array a 0 invece di assegnare 0 a ogni elemento. Per esempio, un array di interi n di cinque elementi può essere reimpostato a 0 con

```
memset(n, 0, 5);
```

Molte architetture hardware hanno istruzioni macchina per copiare o azzerare blocchi di memoria che il compilatore può usare per ottimizzare `memset` e ottenere elevate prestazioni nelle operazioni di azzeramento della memoria. Il programma della Figura 8.25 usa `memset` per copiare 'b' nei primi 7 byte di `string1`.

```
1 // fig08_25.c
2 // Uso della funzione memset
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     char string1[15] = "BBBBBBBBBBBBBBB";
8
9     printf("string1 = %s\n", string1);
10    printf("string1 after memset = %s\n", (char *) memset(string1, 'b', 7));
11 }
```

```
string1 = BBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

Figura 8.25 Uso della funzione `memset`.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni relative alla funzione `memcpy` è *falsa*?
 - a) La funzione copia un numero specificato di byte dall'oggetto a cui punta il suo secondo argomento nell'oggetto a cui punta il suo primo argomento.
 - b) La funzione può ricevere un puntatore a qualunque tipo di oggetto.
 - c) Il risultato di questa funzione è indefinito se i due oggetti sono completamente separati in memoria.
 - d) La funzione è più efficiente di `strcpy` quando si conosce la dimensione della stringa che si sta copiando.

Risposta: c) è *falsa*. In realtà, il risultato è indefinito se i due oggetti si sovrappongono in memoria, ovvero sono parti del medesimo oggetto. In tali casi, usate `memmove`.

2. (*Completare*) Le funzioni di gestione della memoria della libreria per il trattamento delle stringhe effettuano manipolazioni, confronti e ricerche in blocchi di memoria, che le funzioni trattano come _____.

Risposta: array di caratteri.

3. (*Vero/Falso*) La funzione `memmove` copia un numero specificato di byte dall'oggetto puntato dal suo secondo argomento nell'oggetto puntato dal suo primo argomento. La copia è eseguita come se i byte fossero copiati dal secondo argomento in un array di caratteri temporaneo, quindi copiati dall'array temporaneo nel primo argomento. Questo permette di copiare i byte da un blocco di memoria in un'altra parte dello stesso blocco di memoria, anche se le due porzioni si sovrappongono.

Risposta: Vero.

8.10 Altre funzioni della libreria per il trattamento delle stringhe

Le due restanti funzioni della libreria per il trattamento delle stringhe sono `strerror` e `strlen`, che sono riepilogate nella tabella seguente.

Prototipo della funzione	Descrizione della funzione
<code>char *strerror(int errornum);</code>	Mappa <code>errornum</code> in una stringa di testo secondo una modalità specifica per il compilatore e per la località geografica e restituisce la stringa. I numeri di errore sono definiti in <code>errno.h</code> .
<code>size_t strlen(const char *s);</code>	Restituisce la lunghezza della stringa <code>s</code> , ovvero viene restituito il numero di caratteri che precedono il carattere nullo di terminazione.

8.10.1 Funzione `strerror`

La funzione `strerror` riceve un numero di errore e crea una stringa di messaggio di errore. Viene restituito un puntatore alla stringa. Il programma della Figura 8.26 illustra `strerror`.

```

1 // fig08_26.c
2 // Uso della funzione strerror
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     printf("%s\n", strerror(2));
8 }
```

No such file or directory

Figura 8.26 Uso della funzione `strerror`.

8.10.2 Funzione `strlen`

La funzione `strlen` riceve una stringa come argomento e restituisce il numero di caratteri nella stringa (il carattere nullo di terminazione non è incluso nella lunghezza). Il programma della Figura 8.27 illustra la funzione `strlen`.

```

1 // fig08_27.c
2 // Uso della funzione strlen
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *string1 = "abcdefghijklmnopqrstuvwxyz";
8     const char *string2 = "four";
9     const char *string3 = "Boston";
10
11    printf("%s \"%s\" %zu\n%s \"%s\" %zu\n%s \"%s\" %zu\n",
12           "The length of ", string1, strlen(string1),
```

```

13     "The length of ", string2, " is ", strlen(string2),
14     "The length of ", string3, " is ", strlen(string3));
15 }

```

```

The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6

```

Figura 8.27 Uso della funzione `strlen`.

✓ Autovalutazione

1. (*Vero/Falso*) Le stringhe di messaggio di errore restituite dalla funzione `strerror` sono uniformi nelle diverse piattaforme.

Risposta: *Falso*. I messaggi variano a seconda del compilatore e della località geografica.

2. (*Vero/Falso*) La funzione `strlen` riceve una stringa come argomento e restituisce il numero di caratteri nella stringa, incluso il carattere nullo di terminazione.

Risposta: *Falso*. In realtà, il carattere nullo di terminazione non è incluso nella lunghezza.

8.11 Programmazione sicura in C

Funzioni per l'elaborazione sicura di stringhe

Nei precedenti paragrafi “Programmazione sicura in C” di questo libro abbiamo presentato le funzioni del C11 `printf_s` e `scanf_s`. In questo capitolo abbiamo presentato le funzioni `sprintf`, `strcpy`, `strncpy`, `strcat`, `strncat`, `strtok`, `strlen`, `memcpy`, `memmove` e `memset`. Versioni più sicure di queste e di molte altre funzioni di elaborazione di stringhe e di input/output sono descritte dall’Annex K *opzionale* del C11 standard. Se il vostro compilatore C supporta l’Annex K, fareste bene a usare le versioni sicure di queste funzioni. Tra l’altro, le versioni dell’Annex K contribuiscono a evitare gli overflow dei buffer, poiché richiedono ulteriori parametri che specificano il numero di elementi in un array di destinazione e controllano che gli argomenti puntatore siano diversi da `NULL`.

Lettura di input numerici e convalida degli input

È importante convalidare i dati che si inseriscono in un programma. Per esempio, quando chiedete all’utente di inserire un `int` nell’intervallo 1-100 e poi tentate di leggerlo usando `scanf`, è possibile che si presentino diversi problemi. L’utente potrebbe inserire:

- un `int` che è al di fuori dell’intervallo richiesto dal programma (come 102);
- un `int` che è al di fuori dell’intervallo consentito per gli `int` su quel computer (come 8.000.000.000 su una macchina con `int` a 32 bit);
- un valore numerico non intero (come 27,43);
- un valore non numerico (come FOVR).

Potete usare le varie funzioni che avete imparato in questo capitolo per effettuare una convalida completa di un tale input. Per esempio, potreste:

- usare `fgets` per leggere l’input come riga di testo;
- convertire la stringa in un numero usando `strtol` e assicurarvi che la conversione sia riuscita;
- assicurarvi che il valore sia compreso nell’intervallo richiesto.

Per altre informazioni e tecniche per convertire input in valori numerici, consultate la linea guida del CERT INT05-C all’indirizzo <https://wiki.sei.cmu.edu/>.

✓ Autovalutazione

1. (*Completare*) Le funzioni sicure per l'elaborazione di stringhe dell'Annex K sono utili, tra l'altro, per prevenire gli _____ dei buffer, poiché richiedono ulteriori parametri che specificano il numero di elementi in un array di destinazione e controllano che gli argomenti puntatore siano diversi da NULL.

Risposta: overflow.

2. (*Vero/Falso*) È importante convalidare i dati che si inseriscono in un programma. Potete usare diverse funzioni di elaborazione di stringhe e caratteri per una convalida completa degli input. Per esempio, potete usare fgets per leggere l'input come riga di testo, convertire la stringa in un numero usando strtol, assicurarvi che la conversione sia riuscita, poi assicurarvi che il valore sia compreso nell'intervallo richiesto.

Risposta: Vero.

8.12 Riepilogo

Paragrafo 8.2 Nozioni fondamentali su stringhe e caratteri

- I **caratteri** sono i blocchi costituenti fondamentali dei programmi. Ogni programma è composto da una sequenza di caratteri che, quando sono raggruppati insieme in modo sensato, è interpretata dal computer come una serie di istruzioni usate per eseguire un compito.
- Una **costante carattere** è un valore int rappresentato come carattere tra virgolette singole. Il valore di una costante carattere è il valore intero del carattere nell'**insieme di caratteri** della macchina.
- Una **stringa** è una serie di caratteri trattati come unità singola. Una stringa può comprendere lettere, cifre e vari caratteri speciali come +, -, *, / e \$. Le stringhe letterali, ovvero le stringhe costanti, sono scritte in C tra virgolette doppie.
- Una stringa in C è un **array di caratteri** che termina con il **carattere nullo** ('\0').
- A una stringa si accede mediante un **puntatore** al suo primo carattere. Il valore di una stringa è l'**indirizzo** del suo primo carattere.
- Un **array di caratteri** o una **variabile di tipo char *** possono essere inizializzati con una stringa in una definizione.
- Quando viene definito un array di caratteri per contenere una stringa, l'array deve essere grande abbastanza da memorizzare la stringa e il suo carattere nullo di terminazione.
- È possibile memorizzare una stringa in un array usando scanf. La funzione scanf legge i caratteri finché non incontra uno spazio, una tabulazione, un newline o l'indicatore di end-of-file.
- Perché un array di caratteri sia stampato come stringa deve contenere un carattere nullo di terminazione.

Paragrafo 8.3 Libreria per il trattamento dei caratteri

- La funzione **isdigit** determina se il suo argomento è una **cifra** (0-9).
- La funzione **isalpha** determina se il suo argomento è una **lettera maiuscola** (A-Z) o una **lettera minuscola** (a-z).
- La funzione **isalnum** determina se il suo argomento è una **lettera maiuscola** (A-Z), una **lettera minuscola** (a-z) o una **cifra** (0-9).
- La funzione **isxdigit** determina se il suo argomento è una **cifra esadecimale** (A-F, a-f, 0-9).
- La funzione **islower** determina se il suo argomento è una **lettera minuscola** (a-z).
- La funzione **isupper** determina se il suo argomento è una **lettera maiuscola** (A-Z).
- La funzione **toupper** converte una lettera minuscola in una maiuscola e restituisce la lettera maiuscola.
- La funzione **tolower** converte una lettera maiuscola in una minuscola e restituisce la lettera minuscola.

- La funzione **isspace** determina se il suo argomento è uno dei seguenti **caratteri di spaziatura**: ' ' (spazio), '\f', '\n', '\r', '\t' o '\v'.
- La funzione **iscntrl** determina se il suo argomento è uno dei seguenti **caratteri di controllo**: '\t', '\v', '\f', '\a', '\b', '\r' o '\n'.
- La funzione **ispunct** determina se il suo argomento è un **carattere stampabile** diverso da uno spazio, da una cifra o da una lettera.
- La funzione **isprint** determina se il suo argomento è un qualsiasi carattere stampabile, incluso il carattere di spazio.
- La funzione **isgraph** determina se il suo argomento è un carattere stampabile diverso dal carattere di spazio.

Paragrafo 8.4 Funzioni di conversione di stringhe

- La funzione **strtod** converte una sequenza di caratteri che rappresentano un valore in virgola mobile in **double**. Alla locazione specificata dal puntatore all'argomento **char *** viene assegnato il resto della stringa dopo la conversione, o l'intera stringa se non è possibile convertire alcuna porzione della stringa.
- La funzione **strtol** converte una sequenza di caratteri che rappresentano un intero in **long**. Questa funzione opera in maniera identica a **strtod**, ma il terzo argomento specifica la base del valore da convertire.
- La funzione **strtoul** opera in maniera identica a **strtol** ma converte una sequenza di caratteri che rappresentano un intero in **unsigned long int**.

Paragrafo 8.5 Funzioni della libreria standard di input/output

- La funzione **fgets** legge caratteri finché non incontra un carattere newline o l'indicatore di end-of-file. Gli argomenti di **fgets** sono un array di tipo **char**, il numero massimo di caratteri che possono essere letti e lo stream dal quale leggerli. Un carattere nullo ('\0') viene aggiunto in coda all'array dopo il termine della lettura. Se si incontra un newline, esso è incluso nella stringa di input.
- La funzione **putchar** stampa il carattere che riceve come argomento.
- La funzione **getchar** legge un singolo carattere dallo standard input e lo restituisce come un intero. Se incontra l'indicatore di end-of-file, **getchar** restituisce EOF.
- La funzione **puts** riceve una stringa (**char ***) come argomento e la stampa seguita da un carattere newline.
- La funzione **sprintf** usa le stesse specifiche di conversione della funzione **printf** per stampare dati formattati in un array di tipo **char**.
- La funzione **sscanf** usa le stesse specifiche di conversione della funzione **scanf** per leggere dati formattati da una stringa.

Paragrafo 8.6 Funzioni per la manipolazione di stringhe della libreria per il trattamento delle stringhe

- La funzione **strcpy** copia il suo secondo argomento (una stringa) nel suo primo argomento (un array di caratteri). Dovete assicuravi che l'array sia grande abbastanza da memorizzare la stringa e il suo carattere nullo di terminazione.
- La funzione **strncpy** è equivalente a **strcpy**, ma specifica il numero massimo di caratteri da copiare dalla stringa nell'array. Il carattere nullo di terminazione sarà copiato solo se il numero di caratteri da copiare è uno in più della lunghezza della stringa.
- La funzione **strcat** aggiunge la stringa che riceve come secondo argomento (compreso il carattere nullo di terminazione) in coda alla stringa che riceve come primo argomento. Il primo carattere della seconda stringa sostituisce il carattere nullo ('\0') della prima stringa. Dovete assicurarvi che l'array usato per memorizzare la prima stringa sia abbastanza grande da memorizzare sia la prima che la seconda stringa.

- La funzione **strncat** aggiunge in coda alla prima stringa un numero specificato di caratteri della seconda stringa. Un carattere nullo di terminazione viene aggiunto in coda al risultato.

Paragrafo 8.7 Funzioni di confronto della libreria per il trattamento delle stringhe

- La funzione **strcmp** confronta la stringa che riceve come primo argomento con la stringa che riceve come secondo argomento, carattere per carattere. Restituisce 0 se le stringhe sono uguali, restituisce un valore negativo se la prima stringa è minore della seconda e restituisce un valore positivo se la prima stringa è maggiore della seconda.
- La funzione **strncmp** è equivalente a **strcmp**, eccetto per il fatto che **strncmp** confronta un numero specificato di caratteri. Se una delle stringhe è più breve del numero dei caratteri specificato, **strncmp** confronta i caratteri finché non incontra il carattere nullo nella stringa più breve.

Paragrafo 8.8 Funzioni per la ricerca della libreria per il trattamento delle stringhe

- La funzione **strchr** cerca la prima occorrenza di un carattere in una stringa. Se il carattere viene trovato, **strchr** restituisce un puntatore al carattere nella stringa; altrimenti, **strchr** restituisce **NULL**.
- La funzione **strcspn** determina la lunghezza della parte iniziale della stringa nel suo primo argomento che non contiene alcun carattere della stringa nel suo secondo argomento. La funzione restituisce la lunghezza del segmento.
- La funzione **strupr** cerca nel suo primo argomento la prima occorrenza di un qualsiasi carattere che fa parte del suo secondo argomento. Se viene trovato, **strupr** restituisce un puntatore al carattere; altrimenti, **strupr** restituisce **NULL**.
- La funzione **strrchr** cerca l'ultima occorrenza di un carattere in una stringa. Se il carattere viene trovato, **strrchr** restituisce un puntatore al carattere nella stringa; altrimenti, **strrchr** restituisce **NULL**.
- La funzione **strspn** determina la lunghezza della parte iniziale della stringa nel suo primo argomento contenente solo caratteri che fanno parte della stringa nel suo secondo argomento. La funzione restituisce la lunghezza del segmento.
- La funzione **strstr** cerca la prima occorrenza della stringa che riceve come secondo argomento nella stringa che riceve come primo argomento. Se la seconda stringa viene trovata nella prima stringa, viene restituito un puntatore alla locazione della stringa nel primo argomento.
- Una sequenza di chiamate a **strtok** suddivide la stringa che riceve come primo argomento in token che sono separati da caratteri contenuti nella stringa che riceve come secondo argomento. La prima chiamata contiene la stringa da suddividere in token come primo argomento. Le chiamate successive per continuare a spezzare in token la stessa stringa contengono **NULL** come primo argomento. A ogni chiamata viene restituito un puntatore al token corrente. Se non vi sono più token quando la funzione è chiamata, viene restituito **NULL**.

Paragrafo 8.9 Funzioni di gestione della memoria della libreria per il trattamento delle stringhe

- La funzione **memcpy** copia un numero specificato di byte dall'oggetto a cui punta il suo secondo argomento nell'oggetto a cui punta il suo primo argomento. La funzione può ricevere un puntatore a qualunque tipo di oggetto.
- La funzione **memmove** copia un numero specificato di byte dall'oggetto puntato dal suo secondo argomento nell'oggetto puntato dal suo primo argomento. L'operazione di copiatura è eseguita come se i byte fossero copiati dal secondo argomento in un array di caratteri temporaneo e quindi copiati dall'array temporaneo nel primo argomento.
- La funzione **memcmp** confronta il numero specificato di caratteri del suo primo e del suo secondo argomento.

- La funzione `memchr` cerca la prima occorrenza di un byte, rappresentato come un `unsigned char`, all'interno di un numero specificato di byte di un oggetto. Se il byte viene trovato, viene restituito un puntatore al byte, altrimenti viene restituito un puntatore `NULL`.
- La funzione `memset` copia il suo secondo argomento, trattato come un `unsigned char`, in un numero specificato di byte dell'oggetto puntato dal primo argomento.

Paragrafo 8.10 Altre funzioni della libreria per il trattamento delle stringhe

- La funzione `strerror` mappa un numero intero che indica un errore in una stringa di testo secondo una modalità dipendente dal compilatore e dalla posizione geografica. Viene restituito un puntatore alla stringa.
- La funzione `strlen` riceve una stringa come argomento e restituisce il **numero di caratteri** nella stringa (il carattere nullo di terminazione non è incluso nella lunghezza della stringa).

Esercizi di autovalutazione

8.1 Scrivete una singola istruzione per eseguire ognuna delle seguenti operazioni. Supponete che la variabile `c` sia di tipo `char`, le variabili `x`, `y` e `z` siano di tipo `int`, le variabili `d`, `e` e `f` siano di tipo `double`, la variabile `ptr` sia di tipo `char *` e `s1` e `s2` siano array di 100 elementi di tipo `char`.

- a) Convertite il carattere memorizzato nella variabile `c` in una lettera maiuscola. Assegnate il risultato alla variabile `c`.
- b) Determinate se il valore della variabile `c` è una cifra. Usate l'operatore condizionale come mostrato nelle Figure 8.1-8.3 per stampare " is a " o " is not a " quando il risultato viene stampato.
- c) Determinate se il valore della variabile `c` è un carattere di controllo. Usate l'operatore condizionale per stampare " is a " o " is not a " quando il risultato viene stampato.
- d) Leggete una riga di testo dalla tastiera e memorizzatela nell'array `s1`. Non usate `scanf`.
- e) Stampate la riga di testo memorizzata nell'array `s1`. Non usate `printf`.
- f) Assegnate a `ptr` l'indirizzo dell'ultima occorrenza di `c` in `s1`.
- g) Stampate il valore della variabile `c`. Non usate `printf`.
- h) Determinate se il valore di `c` è una lettera. Usate l'operatore condizionale per stampare " is a " o " is not a " quando viene stampato il risultato.
- i) Leggete un carattere dalla tastiera e memorizzatelo nella variabile `c`.
- j) Assegnate a `ptr` la locazione della prima occorrenza di `s2` in `s1`.
- k) Determinate se il valore della variabile `c` è un carattere stampabile. Usate l'operatore condizionale per stampare " is a " o " is not a " quando viene stampato il risultato.
- l) Leggete tre valori `double` nelle variabili `d`, `e` e `f` dalla stringa "1.27 10.3 9.432".
- m) Copiate la stringa memorizzata nell'array `s2` nell'array `s1`.
- n) Assegnate a `ptr` la locazione della prima occorrenza in `s1` di un carattere di `s2`.
- o) Confrontate la stringa in `s1` con la stringa in `s2`. Stampate il risultato.
- p) Assegnate a `ptr` la locazione della prima occorrenza di `c` in `s1`.
- q) Usate `sprintf` per stampare i valori delle variabili intere `x`, `y` e `z` nell'array `s1`. Ogni valore va stampato con un'ampiezza di campo uguale a 7.
- r) Aggiungete 10 caratteri della stringa in `s2` in coda alla stringa in `s1`.
- s) Determinate la lunghezza della stringa in `s1`. Stampate il risultato.
- t) Assegnate a `ptr` la locazione del primo token in `s2`. I token nella stringa `s2` sono separati da virgolette (..).

8.2 Mostrate due metodi differenti per inizializzare l'array `vowel` di tipo `char` con la stringa "AEIOU".

8.3 Che cosa, eventualmente, viene stampato quando viene eseguita ognuna delle seguenti istruzioni in C? Se l'istruzione contiene un errore, descrivetelo e indicate come correggerlo. Presupponete le seguenti definizioni di variabile:

```

char s1[50] = "jack";
char s2[50] = "jill";
char s3[50] = "";

a) printf("%c%s", toupper(s1[0]), &s1[1]);
b) printf("%s", strcpy(s3, s2));
c) printf("%s", strcat(strcat(strcpy(s3, s1), " and "), s2));
d) printf("%zu", strlen(s1) + strlen(s2));
e) printf("%zu", strlen(s3)); // usare s3 dopo l'esecuzione della parte (c)

```

8.4 Trovate l'errore in ognuno dei seguenti segmenti di programma e spiegate come correggerlo.

```

a) char s[10] = "";
   strncpy(s, "hello", 5);
   printf("%s\n", s);
b) printf("%s", 'a');
c) char s[12] = "";
   strcpy(s, "Welcome Home");
d) if (strcmp(string1, string2)) {
      puts("The strings are equal");
}

```

Risposte agli esercizi di autovalutazione

8.1 a) c = toupper(c);
 b) printf("'c'%sdigit\n", c, isdigit(c) ? " is a " : " is not a ");
 c) printf("'c'%scontrol character\n",
 c, iscntrl(c) ? " is a " : " is not a ");
 d) fgets(s1, 100, stdin);
 e) puts(s1);
 f) ptr = strrchr(s1, c);
 g) putchar(c);
 h) printf("'c'%sletter\n", c, isalpha(c) ? " is a " : " is not a ");
 i) c = getchar();
 j) ptr = strstr(s1, s2);
 k) printf("'c'%sprinting character\n",
 c, isprint(c) ? " is a " : " is not a ");
 l) sscanf("1.27 10.3 9.432", "%f%f%f", &d, &e, &f);
 m) strcpy(s1, s2);
 n) ptr = strpbrk(s1, s2);
 o) printf("strcmp(s1, s2) = %d\n", strcmp(s1, s2));
 p) ptr = strchr(s1, c);
 q) sprintf(s1, "%7d%7d%7d", x, y, z);
 r) strncat(s1, s2, 10);
 s) printf("strlen(s1) = %zu\n", strlen(s1));
 t) ptr = strtok(s2, ".");

8.2 char vowel[] = "AEIOU";
 char vowel[] = {'A', 'E', 'I', 'O', 'U', '\0'};

8.3 a) Jack
 b) jill
 c) jack and jill
 d) 8
 e) 13

- 8.4 a) Errore: la funzione `strncpy` non scrive un carattere nullo di terminazione nell'array `s`, perché il suo terzo argomento è uguale alla lunghezza della stringa "hello".
 Correzione: fornite 6 come terzo argomento di `strncpy` o assegnate '\0' a `s[5]`.
- b) Errore: si tenta di stampare una costante carattere come stringa.
 Correzione: usate %c per inviare in uscita il carattere o sostituite 'a' con "a".
- c) Errore: l'array di caratteri `s` non è abbastanza grande da memorizzare il carattere nullo di terminazione.
 Correzione: dichiarate l'array con più elementi.
- d) Errore: la funzione `strcmp` restituisce 0 se le stringhe sono uguali; pertanto, la condizione nell'istruzione `if` risulta falsa e la `printf` non viene eseguita.
 Correzione: confrontate il risultato di `strcmp` con 0 nella condizione.

Esercizi

8.5 (*Test di caratteri*) Scrivete un programma che legga un carattere dalla tastiera e lo testi con ognuna delle funzioni della libreria per il trattamento dei caratteri. Il programma deve stampare il valore restituito da ogni funzione.

8.6 (*Stampa di stringhe con caratteri maiuscoli e minuscoli*) Scrivete un programma che legga una riga di testo e la memorizzi nell'array `char s[100]`. Fate stampare la riga sia con lettere maiuscole sia con lettere minuscole.

8.7 (*Conversione di stringhe in interi per effettuare calcoli*) Scrivete un programma che legga quattro stringhe che rappresentano valori interi, converta le stringhe in interi, sommi i valori e stampi il totale dei quattro valori.

8.8 (*Conversione di stringhe in numeri in virgola mobile per effettuare calcoli*) Scrivete un programma che legga quattro stringhe che rappresentano valori in virgola mobile, converta le stringhe in valori `double`, sommi i valori e stampi il totale dei quattro valori.

8.9 (*Confronto di stringhe*) Scrivete un programma che usi la funzione `strcmp` per confrontare due stringhe inserite dall'utente. Il programma deve stabilire se la prima stringa è minore, uguale o maggiore della seconda.

8.10 (*Confronto di porzioni di stringhe*) Scrivete un programma che usi la funzione `strncmp` per confrontare due stringhe inserite dall'utente. Il programma deve leggere il numero di caratteri da confrontare, quindi stampare se la prima stringa è minore, uguale o maggiore della seconda.

8.11 (*Frasi casuali*) Utilizzate la generazione di numeri casuali per creare frasi in inglese. Il vostro programma deve usare quattro array di puntatori a `char` chiamati `article`, `noun`, `verb` e `preposition`. Create una frase selezionando una parola a caso da ogni array nell'ordine seguente: `article`, `noun`, `verb`, `preposition`, `article` e `noun`. Gli array vanno riempiti come segue: l'array `article` deve contenere gli articoli "the", "a", "one", "some" e "any"; l'array `noun` deve contenere i nomi "boy", "girl", "dog", "town" e "car"; l'array `verb` deve contenere i verbi "drove", "jumped", "ran", "walked" e "skipped"; l'array `preposition` deve contenere le preposizioni "to", "from", "over", "under" e "on".

Concatenate con le parole precedenti ogni parola che viene scelta, usando un array sufficientemente grande da contenere l'intera frase. Separate con spazi le parole. La frase finale deve cominciare con una lettera maiuscola e finire con un punto. Generate 20 frasi di questo tipo. Modificate il vostro programma per produrre un breve racconto contenente diverse di queste frasi. (Che ne dite della possibilità di un programma che scrive tesine a caso?)

8.12 (*Limerick*) Un limerick è una poesiola scherzosa in inglese di cinque versi in cui il primo e il secondo fanno rima con il quinto, mentre il terzo fa rima con il quarto. Usando tecniche simili a quelle sviluppate nell'Esercizio 8.11, scrivete un programma che produca limerick casuali (in inglese o in italiano). Affinare il programma per produrre dei buoni limerick è un problema impegnativo, ma il risultato varrà lo sforzo!

8.13 (*Latino maccheronico*) Scrivete un programma che trasformi frasi della lingua inglese in latino maccheronico (come percepito dalle persone di madrelingua inglese). Il latino maccheronico è una forma di lin-

giaggio codificato usato spesso per divertimento. Esistono molte variazioni nei metodi usati per formare frasi in latino maccheronico. Per semplicità usate l'algoritmo seguente.

Per formare una frase in latino maccheronico da una frase della lingua inglese, spezzate con la funzione `strtok` la frase nelle sue parole costituenti. Per tradurre ogni parola inglese in una parola in latino maccheronico, spostate la prima lettera della parola inglese alla fine della parola e aggiungete le lettere "ay". In questo modo la parola "jump" diventa "umpjay", la parola "the" diventa "hetay" e la parola "computer" diventa "omputercay". Gli spazi tra le parole rimangono tali. Presupponete quanto segue: la frase inglese è formata da parole separate da spazi, non vi sono segni di interpunkzione e tutte le parole hanno due o più lettere. La funzione `printLatinWord` deve stampare ogni parola. [Suggerimento: ogni volta che in una chiamata a `strtok` si trova un token, passate il puntatore al token alla funzione `printLatinWord` e stampate la parola in latino maccheronico. Abbiamo fornito qui regole semplificate per convertire parole in latino maccheronico. Per regole e variazioni più dettagliate, visitate il sito https://en.wikipedia.org/wiki/Pig_latin.]

8.14 (Suddivisione in token di numeri telefonici) Scrivete un programma che legga un numero di telefono come una stringa nella forma (555) 555-5555. Usate la funzione `strtok` per estrarre come singoli token il prefisso, le prime tre cifre e infine le ultime quattro cifre del numero telefonico. Concatenate le sette cifre del numero telefonico in un'unica stringa. Convertite la stringa del prefisso e la stringa del numero telefonico in interi, poi stampate entrambi.

8.15 (Stampa di una frase all'inverso) Scrivete un programma che legga una riga di testo, la spezzi in token con la funzione `strtok` e invii in uscita i token in ordine inverso.

8.16 (Ricerca di sottostringhe) Scrivete un programma che legga dalla tastiera una riga di testo e una stringa da ricercare. Usando la funzione `strstr`, cercate nella riga di testo la prima occorrenza della stringa da ricercare. Assegnate l'indirizzo alla variabile `searchPtr` di tipo `char *`. Se la stringa viene trovata, stampate il resto della riga di testo cominciando con tale stringa. Poi usate di nuovo `strstr` per localizzare la successiva occorrenza della stringa nella riga di testo. Se questa viene trovata, stampate il resto della riga di testo cominciando con la seconda occorrenza. [Suggerimento: la seconda chiamata a `strstr` deve contenere `searchPtr + 1` come suo primo argomento.]

8.17 (Conteggio delle occorrenze di una sottostringa) Scrivete un programma basato sull'Esercizio 8.16 che legga diverse righe di testo e una stringa di ricerca e che usi la funzione `strstr` per determinare il totale delle volte in cui la stringa ricorre nelle righe di testo. Stampate il risultato.

8.18 (Conteggio delle occorrenze di un carattere) Scrivete un programma che legga diverse righe di testo e un carattere da ricercare e che usi la funzione `strchr` per determinare il totale delle volte in cui il carattere compare nelle righe di testo.

8.19 (Conteggio delle lettere dell'alfabeto in una stringa) Scrivete un programma basato sul programma dell'Esercizio 8.18 che legga diverse righe di testo e che usi la funzione `strchr` per determinare il totale delle volte in cui ogni lettera dell'alfabeto compare nelle righe di testo. Le lettere maiuscole e minuscole vanno contate insieme. Memorizzate i totali per ogni lettera in un array e stampate i valori in formato tabellare dopo che sono stati determinati i totali.

8.20 (Conteggio del numero di parole in una stringa) Scrivete un programma che legga diverse righe di testo e che usi `strtok` per contare il numero totale di parole. Supponete che le parole siano separate o da spazi o da caratteri newline.

8.21 (Mettere in ordine alfabetico una lista di stringhe) Usate le funzioni per il confronto di stringhe e le tecniche per ordinare gli array per scrivere un programma che metta in ordine alfabetico una lista di stringhe. Usate i nomi di 10 o 15 città come dati per il vostro programma.

8.22 L'Appendice B mostra le rappresentazioni con codici numerici dell'insieme di caratteri ASCII. Studiate l'Appendice B e poi stabilite se ognuna delle seguenti affermazioni è vera o falsa.

- La lettera "A" viene prima della lettera "B".
- La cifra "9" viene prima della cifra "0".

- c) I simboli comunemente usati per l'addizione, la sottrazione, la moltiplicazione e la divisione vengono tutti prima di una qualunque cifra.
- d) Le cifre vengono prima delle lettere.
- e) Se un programma di ordinamento dispone le stringhe in una sequenza crescente, porrà il simbolo per una parentesi destra prima del simbolo per una parentesi sinistra.

8.23 (Stringhe che cominciano con "b") Scrivete un programma che legga una serie di stringhe e stampi solo quelle che cominciano con la lettera "b".

8.24 (Stringhe che finiscono con "ed") Scrivete un programma che legga una serie di stringhe e stampi solo quelle che finiscono con le lettere "ed".

8.25 (Stampa di lettere per vari codici ASCII) Scrivete un programma che legga un codice ASCII e stampi il carattere corrispondente.

8.26 (Scrivete le vostre funzioni per il trattamento dei caratteri) Usando come guida la tabella di caratteri ASCII nell'Appendice B, scrivete le vostre versioni delle funzioni per il trattamento dei caratteri del Paragrafo 8.3.

8.27 (Scrivete le vostre funzioni di conversione di stringhe) Scrivete le vostre versioni delle funzioni del Paragrafo 8.4 per convertire stringhe in numeri.

8.28 (Scrivete le vostre funzioni per copiare e concatenare stringhe) Scrivete due versioni di ognuna delle funzioni per copiare e concatenare stringhe del Paragrafo 8.6. La prima versione deve usare l'indicizzazione di array e la seconda i puntatori e l'aritmetica dei puntatori.

8.29 (Scrivete le vostre funzioni di confronto di stringhe) Scrivete due versioni di ogni funzione di confronto di stringhe della Figura 8.13. La prima versione deve usare l'indicizzazione di array e la seconda i puntatori e l'aritmetica dei puntatori.

8.30 (Scrivete la vostra funzione per il calcolo della lunghezza di stringhe) Scrivete due versioni della funzione `strlen` della Figura 8.27. La prima versione deve usare l'indicizzazione di array e la seconda i puntatori e l'aritmetica dei puntatori.

Paragrafo speciale: esercizi avanzati di manipolazione di stringhe

Gli esercizi precedenti sono adeguati al testo e ideati per verificare la comprensione da parte del lettore dei concetti fondamentali della manipolazione di stringhe. Questo paragrafo contiene problemi intermedi e avanzati che troverete impegnativi ma divertenti e di diversi gradi di difficoltà. Alcuni richiedono un'ora o due di programmazione, altri sono utili per attività di laboratorio che potrebbero richiedere due o tre settimane di studio e di implementazione. Alcuni sono progetti impegnativi a lungo termine.

8.31 (Analisi di testi) Le funzionalità orientate alla manipolazione di stringhe consentono alcuni approcci alquanto interessanti all'analisi degli scritti di grandi autori. Molta attenzione si è concentrata sul fatto se William Shakespeare sia mai vissuto. Alcuni studiosi ritengono che ci siano prove sufficienti che Christopher Marlowe abbia in realtà composto i capolavori attribuiti a Shakespeare. I ricercatori hanno usato i computer per trovare somiglianze nelle opere di questi due autori. Questo esercizio esamina tre metodi per analizzare testi con un computer.

- a) Scrivete un programma che legga diverse righe di testo e stampi una tabella che indichi il numero di volte che ogni lettera dell'alfabeto ricorre nel testo. Per esempio, la frase seguente contiene una "a", due "b", nessuna "c" e così via:

To be, or not to be: that is the question:

- b) Scrivete un programma che legga diverse righe di testo e stampi una tabella che indichi il numero delle parole di una lettera, delle parole di due lettere, di tre lettere ecc. che compaiono nel testo. Per esempio, la frase

Whether 'tis nobler in the mind to suffer

contiene quanto illustrato nella tabella seguente.

Lunghezza delle parole	Occorrenza
1	0
2	2
3	1
4	2 (compreso 'tis)
5	0
6	2
7	1

- c) Scrivete un programma che legga diverse righe di testo e stampi una tabella che indichi il numero di volte che ogni parola diversa ricorre nel testo. Il programma deve includere le parole nella tabella nello stesso ordine in cui compaiono nel testo. Per esempio, le righe

To be, or not to be: that is the question:

Whether 'tis nobler in the mind to suffer

contengono le parole "to" tre volte, "be" due volte, "or" una volta, e così via.

- 8.32 (*Stampa di date in vari formati*) Nelle corrispondenze d'affari le date sono comunemente stampate in diversi formati differenti. Due dei formati più comuni sono

07/21/2003 e July 21, 2003

Scrivete un programma che legga una data nel primo formato e la stampi nel secondo.

- 8.33 (*Protezione di assegni*) I computer sono usati frequentemente in sistemi per la scrittura di assegni, come in applicazioni di libri paga e di conti fornitori. Circolano molte storie sugli assegni paga settimanali stampati (per errore) con importi in eccesso di 1 milione di dollari. Vengono stampati strani importi da sistemi computerizzati per la scrittura di assegni a causa di errori umani e/o di malfunzionamenti delle macchine. I progettisti di sistemi, naturalmente, fanno ogni sforzo per inserire controlli nei loro sistemi al fine di evitare che siano emessi assegni sbagliati.

Un altro problema serio è quello dell'alterazione intenzionale dell'importo di assegni da parte di qualcuno che intende incassarli in modo fraudolento. Per evitare che un importo in dollari venga alterato, la maggior parte dei sistemi computerizzati per la scrittura di assegni impiega una tecnica chiamata *protezione di assegni*.

Gli assegni destinati a essere stampati da un computer contengono un numero fisso di spazi in cui il computer può stampare un importo. Supponete che un assegno paga contenga nove spazi in cui il computer è tenuto a stampare l'importo di una paga settimanale. Se l'importo è grande, allora tutti quei nove spazi verranno riempiti, come in questo caso:

11.230.60 (importo dell'assegno)

123456789 (numeri di posizione)

D'altra parte, se l'importo è inferiore a 1.000 dollari, saranno normalmente lasciati vuoti diversi spazi; per esempio,

99.87

123456789

contiene quattro spazi vuoti. Se un assegno viene stampato con spazi vuoti, è più facile alterarne l'importo. Per evitare tale alterazione, molti sistemi per la scrittura di assegni inseriscono *asterischi iniziali* per proteggere l'importo come segue:

****99.87

123456789

Scrivete un programma che legga un importo in dollari da stampare su un assegno e che poi stampi l'importo nel formato di assegno protetto con asterischi iniziali, se necessario. Supponete che per stampare un importo siano disponibili nove spazi.

8.34 (Scrittura dell'equivalente in lettere dell'importo di un assegno) Continuando l'analisi dell'esercizio precedente, un comune metodo di sicurezza richiede che l'importo di un assegno venga scritto sia in numeri che in lettere. Anche se qualcuno fosse capace di alterare l'importo numerico dell'assegno, è estremamente difficile cambiare l'importo scritto in lettere. Scrivete un programma che legga l'importo numerico di un assegno e ne scriva l'equivalente in lettere. Per esempio, l'importo 52,43 deve essere scritto come

FIFTY TWO and 43/100

8.35 (Progetto: un programma di conversione metrica) Scrivete un programma per assistere l'utente in operazioni di conversione metrica. Il vostro programma deve permettere all'utente di specificare i nomi delle unità come stringhe (cioè centimetri, litri, grammi, ecc. per il sistema metrico, e pollici, quarti di gallone, libbre, ecc. per il sistema inglese) e deve rispondere a semplici domande come

"Quanti pollici vi sono in 2 metri?"

"Quanti litri vi sono in 10 quarti di gallone?"

Il vostro programma deve riconoscere le conversioni non valide. Per esempio, la domanda seguente è del tutto priva di senso, perché "piedi" riguarda un'unità di lunghezza, mentre "chilogrammi" riguarda un'unità di massa.

"Quanti piedi vi sono in 5 chilogrammi?"

8.36 (Cucinare con ingredienti più sani) Negli Stati Uniti l'obesità sta aumentando a un ritmo allarmante. Controllate la pagina web di Centers for Disease Control and Prevention (CDC) all'indirizzo www.cdc.gov/obesity/data/index.html, che contiene dati e statistiche sull'obesità negli Stati Uniti. Aumentando l'obesità, aumentano anche i casi riguardanti problemi a essa correlati (es. malattie cardiache, pressione sanguigna alta, colesterolo alto, diabete di tipo 2). Scrivete un programma che aiuti l'utente a scegliere gli ingredienti quando cucina e che aiuti gli allergici a certi cibi (es. noci, glutine) a trovarne di alternativi. Il programma deve leggere una ricetta fornita dall'utente e consigliare ingredienti sostitutivi più sani. Per semplicità, il vostro programma deve presupporre che la ricetta non contenga abbreviazioni per le misure quali cucchiaini da tè, tazze e cucchiaini, e usi cifre numeriche per le quantità (es. 1 uovo, 2 tazze) invece che lettere (un uovo, due tazze). Alcune comuni sostituzioni con alimenti alternativi sono mostrate nella tabella seguente. Il vostro programma deve stampare un avviso come "Consultate sempre il vostro medico prima di apportare modifiche significative alla vostra dieta".

Ingrediente	Sostituzione
1 tazza di panna acida	1 tazza di yogurt
1 tazza di latte	1/2 tazza di latte condensato senza zucchero e 1/2 tazza di acqua
1 cucchiaino di succo di limone	1/2 cucchiaino di aceto
1 tazza di zucchero	1/2 tazza di miele, 1 tazza di melassa o 1/4 di tazza di nettare di agave
1 tazza di burro	1 tazza di margarina o di yogurt
1 tazza di farina	1 tazza di farina di segale o di farina di riso
1 tazza di maionese	1 tazza di fiocchi di latte o 1/8 di tazza di maionese e 7/8 di tazza di yogurt
1 uovo	2 cucchiaini di amido di mais, fecola di maranta o amido di patate o 2 albumi o 1/2 di una grossa banana (schiacciata)
1 tazza di latte	1 tazza di latte di soia
1/4 di tazza di olio	1/4 di tazza di salsa di mele
pane bianco	pane integrale

Il vostro programma deve inoltre prendere in considerazione il fatto che il rapporto delle sostituzioni non è sempre di uno a uno. Per esempio, se la ricetta di una torta richiede tre uova, essa potrebbe invece usare ragionevolmente sei albumi. È possibile ottenere dati di conversione per le misure e i cibi sostitutivi da numerosi siti web. Il vostro programma deve considerare le preoccupazioni dell'utente per la sua salute, come colesterolo alto, pressione alta, aumento di peso, allergia al glutine e così via. Per il colesterolo alto il programma deve suggerire alimenti alternativi alle uova e ai latticini; se l'utente desidera perdere peso, vanno consigliati cibi poco calorici al posto di ingredienti come lo zucchero.

8.37 (Filtro anti-spam) Lo spam (o posta indesiderata) comporta un costo per le organizzazioni statunitensi di miliardi di dollari all'anno in software per la prevenzione, dispositivi, risorse di rete, larghezza di banda e perdita di produttività. Cercate online alcuni dei più comuni messaggi di spam inviati per e-mail e controllate la vostra cartella di posta indesiderata. Create una lista di 30 parole e frasi che si trovano comunemente nei messaggi di spam. Scrivete un programma in cui l'utente inserisca un messaggio di posta elettronica. Il vostro programma deve leggere il messaggio e memorizzarlo in un array di caratteri di grandi dimensioni e assicuratevi che il programma non tenti di inserire caratteri oltre la fine dell'array. Poi analizzate il messaggio per verificare la presenza di ognuna delle 30 parole chiave o frasi. Per ogni occorrenza di una di queste parole o frasi nel messaggio, aggiungete un punto al "punteggio spam" del messaggio. Successivamente, valutate la probabilità che il messaggio sia spam in base al numero di punti ricevuti.

8.38 (Linguaggio per SMS) Lo Short Message Service (SMS) è un servizio di comunicazioni fra telefoni cellulari che permette di inviare messaggi di testo di un massimo di 160 caratteri. Con la proliferazione in tutto il mondo dell'uso dei cellulari, l'SMS è usato in molte nazioni in via di sviluppo per fini politici (es. dare voce a opinioni e all'opposizione), per riferire notizie su disastri naturali, e così via. Poiché la lunghezza dei messaggi via SMS è limitata, si usa spesso il linguaggio SMS: abbreviazioni di parole e frasi comuni nei messaggi di testo per cellulari, e-mail, messaggi istantanei, ecc. Per esempio, "ti voglio bene" diventa TVB nel linguaggio SMS. Cercate online linguaggi SMS. Scrivete un programma che faccia inserire all'utente un messaggio con un linguaggio SMS e che poi lo traduca nella lingua che preferite. Fornite anche un meccanismo per tradurre un testo scritto nella vostra lingua nel linguaggio SMS. Un problema potenziale è costituito dal fatto che un'abbreviazione in SMS potrebbe espandersi in una varietà di frasi. Per esempio, TVB (come usato in precedenza) potrebbe stare anche per "tutto va bene", "Ti Voglio Bere" (campagna per l'uso consapevole dell'acqua), ecc.

8.39 (Neutralità di genere) Nell'Esercizio 1.6, avete cercato di eliminare il sessismo in tutte le forme di comunicazione e avete descritto l'algoritmo che usereste per leggere da cima a fondo un paragrafo di testo e sostituire i termini specifici di genere con equivalenti neutrali. Create un programma che legga un paragrafo di testo, quindi sostituisca i termini specifici di genere con quelli neutrali. Stampate il testo neutrale che ne risulta.

Un progetto impegnativo di manipolazione di stringhe

8.40 (Progetto: un generatore di cruciverba) Molte persone in varie occasioni si impegnano nella risoluzione di cruciverba, ma poche hanno mai tentato di crearne uno. Creare un cruciverba è un compito difficile. Qui è proposto come un progetto di manipolazione di stringhe che richiede uno sforzo e un grado di sofisticatezza notevoli. È necessario risolvere molti problemi per far funzionare anche il più semplice programma generatore di parole crociate. Per esempio, come si può rappresentare la griglia di un cruciverba all'interno del computer? Si devono usare serie di stringhe, o forse array bidimensionali? Occorre una fonte di parole (ossia un dizionario computerizzato) a cui il programma possa fare direttamente riferimento. In quale forma vanno memorizzate queste parole per facilitare le complesse manipolazioni richieste dal programma? Il lettore veramente ambizioso vorrà creare la porzione delle "definizioni" del cruciverba in cui sono stampati per gli appassionati del gioco i brevi suggerimenti riguardo a ogni parola "orizzontale" e "verticale". Ma già soltanto stampare una versione del cruciverba vuoto non è un problema semplice.

Pqyoaf X Nylfomigrob Qwbbfmh Mndogvk: Rboqlrut yua Boklnxhmywex

8.41 (Pqyoaf X Nylfomigrob: Cuzqvpcxo vlk Adzdujcjjl) Avrete senza dubbio notato che il titolo di questa sezione e il titolo dell'esercizio sembrano incomprensibili. Non si tratta di un errore! In questo esercizio, continuiamo a porre attenzione al tema della sicurezza introducendo la crittografia. Creerete funzioni che implementano un **cifrario a chiave segreta di Vigenère**.³⁴ Dopo aver imparato a crittografare e decrittografare un vostro testo, potete usare la funzione `decrypt` con la nostra chiave segreta per decriptare i titoli qui sopra.

Crittografia

 La crittografia è stata usata da migliaia di anni^{5,6} ed è di importanza fondamentale nel nostro mondo moderno interconnesso. Ogni giorno, dietro le quinte, la crittografia viene usata per assicurare la privacy e la sicurezza delle vostre comunicazioni via Internet. Per esempio, la maggior parte dei siti web (incluso `deitel.com`) usano oggi il protocollo HTTPS (*Hypertext Transfer Protocol Secure*) per crittografare e decrittografare le interazioni web.

Cifrario di Cesare

Giulio Cesare ha usato un semplice **cifrario a sostituzione** per crittografare le comunicazioni militari.⁷ Nota come il **cifrario di Cesare**, questa tecnica sostituisce ciascuna lettera in un messaggio con la lettera che si trova tre posizioni dopo nell'alfabeto. Quindi, A è sostituita con D, B con E, C con F, ... X con A, Y con B e Z con C. Pertanto, il testo non crittografato

Caesar Cipher

verrebbe crittografato come

Fdhvdu Flskhu

Il testo crittografato viene chiamato **testo cifrato**. Il testo non crittografato viene chiamato **testo in chiaro**.

Sperimentare con i cfrari

Potete divertirvi con il cfrario di Cesare e molti altri algoritmi di crittografia visitando il sito:

<https://cryptii.com/pipes/caesar-cipher>

che è un'implementazione online del progetto open source cryptii:

<https://github.com/cryptii/cryptii>

Nel sito `cryptii.com`, potete inserire testo in chiaro, scegliere quale cfrario usare, specificare le impostazioni del cfrario e vedere il testo cifrato risultante.

Cifrario di Vigenère

È relativamente semplice decrittografare i cfrari a sostituzione semplice come il cfrario di Cesare. Per esempio, "e" è la lettera usata con più frequenza nella lingua inglese; potete quindi studiare un testo cifrato in inglese assumendo che il carattere che appare con più frequenza sia probabilmente una "e".

Il cfrario a chiave segreta di Vigenère usa lettere del testo in chiaro e una chiave segreta per localizzare i caratteri sostitutivi in 26 cfrari di Cesare (uno per ogni lettera dell'alfabeto). Questi 26 cfrari formano un array bidimensionale 26-per-26 chiamato **tavola (o quadrato) di Vigenère**:

-
3. "Crypto Corner—Vigenère Cipher." Accesso 23 dicembre 2020. <https://crypto.interactive-maths.com/vigenegravere-cipher.html>.
 4. "Vigenère cipher." Accesso 23 dicembre 2020. https://en.wikipedia.org/wiki/Vigenère_cipher.
 5. "Cryptography." Accesso 23 dicembre 2020. https://en.wikipedia.org/wiki/Cryptography#History_of_cryptography_and_cryptanalysis.
 6. Binance Academy, "History of Cryptography." Accesso 23 dicembre 2020. <https://www.binance.vision/security/history-of-cryptography>.
 7. "Caesar Cipher." Accesso 23 dicembre 2020. https://en.wikipedia.org/wiki/Caesar_cipher.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	

Cercate le sostituzioni usando le lettere in grassetto che fungono da etichette di righe e colonne.

Requisiti della chiave segreta

Nel caso del cifrario di Vigenère qui descritto, la chiave segreta deve contenere solamente lettere. Così come avviene per le password, anche la chiave segreta non deve essere facile da indovinare. Per creare il testo cifrato nei titoli all'inizio di questo esercizio, abbiamo usato come chiave i seguenti 11 caratteri scelti casualmente:

XMWUJBVYHXZ

La vostra chiave può avere il numero di caratteri che volete. La persona che decripterà il testo cifrato **dove** **essere a conoscenza della chiave segreta usata per creare il testo cifrato**.⁸ Presumibilmente, l'avrete fornita in anticipo, se possibile di persona. La chiave segreta, ovviamente, deve essere protetta con attenzione.

L'algoritmo di crittografia del codice di Vigenère

Per vedere come opera il cifrario di Vigenère, usiamo la chiave “XMWUJBVYHXZ” e crittografiamo la stringa di testo in chiaro:

Welcome to encryption

Le implementazioni di crittografia e decrittografia da noi usate mantengono invariate le lettere minuscole e maiuscole del testo in chiaro originale. Le lettere maiuscole nel testo in chiaro rimangono maiuscole nel testo cifrato e viceversa; lo stesso vale per le lettere minuscole. Abbiamo scelto di passare i caratteri diversi dalle lettere (come spazi, cifre e segni di punteggiatura) così come sono dal testo in chiaro al testo cifrato e viceversa.

8. Esistono molti siti web che mettono a disposizione decodificatori del cifrario di Vigenère che tentano di decriptare un testo cifrato senza l'ausilio della chiave segreta originale. Ne abbiamo provati diversi, ma nessuno è riuscito a restituire il nostro testo originale.

Per prima cosa, ripetiamo la chiave segreta fino a quando la lunghezza non corrisponde a quella del testo in chiaro:

Testo in chiaro:

Testo che ripete la chiave:

W	e	l	c	o	m	e	t	o	e	n	c	r	y	p	t	i	o	n
X	M	W	U	J	B	V	Y	H	X	Z	X	M	W	U	J	B	V	Y

Nello schema soprastante, abbiamo evidenziato in grigio chiaro la chiave segreta, e poi in grigio scuro le otto lettere ripetute della chiave segreta.

Iniziamo la crittografia usando la prima lettera nel testo che ripete la chiave ("X") per selezionare una riga nella tavola di Vigenère e usando la prima lettera nel testo in chiaro ("W") per selezionare una colonna. L'intersezione tra quella riga e quella colonna (evidenziata qui di seguito) contiene la lettera che deve sostituire la "W" nel testo cifrato, in questo caso "T":

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Si prosegue con lo stesso procedimento per ogni coppia di lettere della chiave segreta e del testo in chiaro:

Testo in chiaro:

Testo che ripete la chiave:

Testo cifrato:

W	e	l	c	o	m	e	t	o	e	n	c	r	y	p	t	i	o	n
X	M	W	U	J	B	V	Y	H	X	Z	X	M	W	U	J	B	V	Y
T	q	h	w	x	n	z	r	v	b	m	z	d	u	j	c	j	j	l

Decrittografia con il cifrario di Vigenère

Il processo di decrittografia restituisce il testo originale in chiaro a partire dal testo cifrato. È simile a quello appena descritto e richiede la medesima chiave segreta usata per crittografare il testo. L'algoritmo di decrittografia, come quello di crittografia, itera sulle lettere della chiave segreta. Quindi, ripetiamo ancora la chiave fino a quando la lunghezza non corrisponde a quella del testo cifrato:

Testo cifrato:

T	q	h	w	x	n	z	r	v	b	m	z	d	u	j	c	j	j	l
X	M	W	U	J	B	V	Y	H	X	Z	X	M	W	U	J	B	V	Y

Testo che ripete la chiave:

Iniziamo la decrittografia usando la prima lettera nel testo che ripete la chiave ("X") per selezionare una riga nella tavola di Vigenère. Localizziamo poi all'interno della riga la prima lettera del testo cifrato ("T"). Infine, sostituiamoci la lettera del testo cifrato con la lettera del testo in chiaro che si trova in cima a quella colonna ("W"), come evidenziato nella seguente tavola di Vigenère:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	Y	

Si prosegue con lo stesso procedimento per ogni coppia di lettere della chiave segreta e del testo cifrato:

Testo cifrato:

T	q	h	w	x	n	z	r	v	b	m	z	d	u	j	c	j	j	l
X	M	W	U	J	B	V	Y	H	X	Z	X	M	W	U	J	B	V	Y
W	e	l	c	o	m	e	t	o	e	n	c	r	y	p	t	i	o	n

Testo che ripete la chiave:

Testo in chiaro:

Implementazione del cifrario di Vigenère

In questo esercizio, implemetterete il codice del vostro cifrario di Vigenère nel file `cipher.c`. Questo file di codice sorgente deve contenere i seguenti elementi:

- La **funzione checkKey** riceve una chiave segreta in forma di stringa e restituisce `true` se la stringa contiene solamente lettere. Altrimenti, la funzione restituisce `false`, e in tal caso non può essere utilizzata con l'algoritmo del cifrario di Vigenère. Questa funzione viene chiamata dalle funzioni `encrypt` e `decrypt` descritte qui sotto.
- La **funzione getSubstitution** riceve un carattere della chiave segreta, un carattere da una stringa di testo in chiaro o di testo cifrato e un tipo `bool` che indica se il carattere nel secondo argomento è da crittografare (`true`) oppure da decrittografare (`false`). Questa funzione viene chiamata dalle funzioni `encrypt` e `decrypt` (descritte qui sotto) per eseguire l'algoritmo di crittografia o decrittografia del cifrario di Vigenère per un carattere. La funzione contiene la tavola di Vigenère in forma di array di caratteri `static const bidimensionale 26-per-26`.
- La **funzione encrypt** riceve una stringa che contiene il **testo in chiaro da crittografare**, un array di caratteri nel quale scrivere il **testo crittografato**, e la **chiave segreta**. La funzione itera sui caratteri del testo in chiaro. Per ciascuna lettera, la funzione `encrypt` chiama la funzione `getSubstitution`, passandole il carattere corrente della chiave segreta, la lettera da crittografare e `true`. La funzione `getSubstitution` esegue quindi l'algoritmo di crittografia del cifrario di Vigenère per quella lettera e ne restituisce l'equivalente nel testo cifrato.
- La **funzione decrypt** riceve una stringa che contiene il **testo cifrato da decrittografare**, un array di caratteri nel quale scrivere il **testo in chiaro risultante**, e la **chiave segreta** usata per creare il testo cifrato. La funzione itera sui caratteri del testo cifrato. Per ciascuna lettera, la funzione `decrypt` chiama la funzione `getSubstitution`, passandole il carattere corrente della chiave segreta, la lettera da decrittografare e `false`. La funzione `getSubstitution` esegue quindi l'algoritmo di decrittografia del cifrario di Vigenère per quella lettera e ne restituisce l'equivalente nel testo in chiaro.

Ulteriori file da creare

Oltre al file `cipher.c`, create i seguenti file di codice:

- `cipher.h` che contiene i **prototipi delle funzioni encrypt e decrypt**;
- `cipher_test.c`, che include (`#include`) "cipher.h" e usa le vostre funzioni `encrypt` e `decrypt` per crittografare e decrittografare testo.

`cipher_test.c`

Nella vostra applicazione, effettuate le seguenti attività:

1. Richiedete e inserite come input una **frase in chiaro da crittografare** e una **chiave segreta** composta solamente da lettere, chiamate la funzione `encrypt` per creare il **testo cifrato**, quindi stampatelo. Usate la nostra chiave segreta XMWUJBVYHZ (che vi consentirà di decifrare il testo incomprensibile all'inizio dell'esercizio).
2. Usate la vostra **funzione decrypt** e la **chiave segreta** che avete inserito nel *Punto 1* per **decrittografare il testo cifrato** che avete appena creato. Stampate il **testo in chiaro** risultante per assicurarvi che la vostra **funzione decrypt** abbia operato in modo corretto.
3. Richiedete e inserite come input il titolo cifrato del paragrafo che precede questo esercizio oppure il titolo cifrato dell'esercizio stesso. Usate poi la vostra **funzione decrypt** e il **testo della chiave segreta** che avete inserito nel *Punto 1* per **decrittografare il testo cifrato**.

Dovete come sempre assicurarvi che gli array di caratteri nei quali scrivete il testo crittografato o decrittografato siano di dimensioni abbastanza grandi da contenere il testo e il carattere nullo di terminazione.

Dopo aver reso operativi i vostri algoritmi di crittografia e decrittografia del cifrario di Vigenère, potete divertirvi con i vostri amici a spedire e ricevere messaggi cifrati. Prestate molta attenzione alla sicurezza quando trasmettete la vostra chiave segreta alla persona che la userà per decrittografare i vostri messaggi cifrati.

Compilazione del vostro codice

In Visual C++ e Xcode, aggiungete semplicemente tutti e tre i file al vostro progetto, poi compilate ed eseguite il codice. Per il compilatore GNU gcc, eseguite il comando seguente dalla cartella contenente i file `cipher.c`, `cipher.h` e `cipher_test.c`:

```
gcc -std=c18 -Wall cipher.c cipher_test.c -o cipher_test
```

Questo creerà il comando `cipher_test`, che potete eseguire con `./cipher_test`.

Punti deboli della crittografia a chiave segreta: un cenno alla crittografia a chiave pubblica

La crittografia e la decrittografia a chiave segreta hanno un difetto: il testo cifrato è sicuro solo nella misura in cui è sicura la chiave segreta. Il testo cifrato può essere decrittografato da chiunque scopra o sottragga la chiave. Introdurremo nel prossimo esercizio la **crittografia a chiave pubblica**. Questa tecnica esegue la crittografia con una chiave pubblica nota a tutti quelli che vogliono mandare un messaggio segreto a un destinatario particolare. La chiave pubblica può essere usata per crittografare i messaggi ma non per decrittografarli. I messaggi possono essere decrittografati unicamente con un'abbinata chiave privata conosciuta solo dal destinatario, che è pertanto molto più sicura rispetto alla chiave segreta della crittografia a chiave segreta. Nell'esercizio del prossimo caso pratico, esplorrete la crittografia a chiave pubblica.

Crittografia e potenza di elaborazione

In teoria, dovrebbe essere impossibile “rompere” un testo cifrato, cioè ricavare il testo in chiaro dal testo cifrato *senza* la chiave di decrittografia. Tale obiettivo, per vari motivi, è irrealizzabile. Per questo gli ideatori di sistemi crittografici si accontentano di renderli estremamente difficili da rompere. Il problema è che i moderni computer sempre più potenti riescono a rompere la maggior parte dei sistemi crittografici usati negli ultimi decenni.

La crittografia è alle origini di criptovalute come i Bitcoin.⁹ I computer quantistici potranno raggiungere una potenza eccezionale e mettere così a rischio i sistemi crittografici e le criptovalute.^{10,11} Gli esperti di criptovalute stanno lavorando per affrontare queste sfide.^{12,13,14}

8.42 (Modificare il cifrario di Vigenère in modo che supporti tutti i caratteri ASCII) L'implementazione del cifrario di Vigenère nell'Esercizio 8.41 può crittografare e decrittografare solamente le lettere dalla A alla Z. Tutti gli altri caratteri vengono semplicemente passati così come sono. Modificate la vostra implementazione per fare in modo che supporti l'insieme di caratteri ASCII completo presentato nell'Appendice B.

Programmazione sicura in C

Caso pratico: crittografia a chiave pubblica

8.43 (Crittografia a chiave pubblica RSA^{15,16,17}) Nel caso pratico precedente vi siete accostati alla crittografia a chiave segreta. Il testo in chiaro del mittente viene crittografato con una chiave segreta per formare testo cifrato. Il destinatario usa la *stessa* chiave per decodificare il testo cifrato e ricreare il testo originale in chiaro: questa viene detta **crittografia simmetrica**. La crittografia a chiave segreta ha un problema: il testo cifrato è sicuro solo nella misura in cui è sicura la chiave segreta, e numerose copie di quella chiave sono “in giro”. Con

9. “Cryptocurrency,” Accesso 25 dicembre 2020. <https://www.investopedia.com/terms/c/cryptocurrency.asp>.

10. “The Impact of Quantum Computing on Present Cryptography,” Accesso 25 dicembre 2020. <https://arxiv.org/pdf/1804.00200.pdf>.

11. “Quantum Computing and its Impact on Cryptography,” Accesso 25 dicembre 2020. <https://www.cryptomathic.com/news-events/blog/quantum-computing-and-its-impact-on-cryptography>.

12. “How Should Crypto Prepare for Google’s ‘Quantum Supremacy?’” Accesso 25 dicembre 2020. <https://www.coindesk.com/how-should-crypto-prepare-for-googles-quantumsupremacy>.

13. “Here’s Why Quantum Computing Will Not Break Cryptocurrencies,” Accesso 25 dicembre 2020. <https://www.forbes.com/sites/rogerhuang/2020/12/21/heres-why-quantumcomputing-will-not-break-cryptocurrencies/>.

14. “How the Crypto World Is Preparing for Quantum Computing, Explained,” Accesso 25 dicembre 2020. <https://cointelegraph.com/explained/how-the-crypto-world-is-preparing-for-quantum-computing-explained>.

15. “RSA (cryptosystem),” Accesso 6 gennaio 2021. [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem)).

16. “RSA Algorithm,” Accesso 6 gennaio 2021. https://simple.wikipedia.org/wiki/RSA_algorithm.

17. “PKCS #1: RSA Cryptography Specifications Version 2.2,” Accesso 8 gennaio 2021. <https://tools.ietf.org/html/rfc8017>.

lo scopo di risolvere questo problema, è stato proposto lo schema Diffie-Hellman¹⁸, un sistema di crittografia a chiave pubblica.

Nell'esercizio di questo caso pratico, esamineremo fase per fase l'**algoritmo RSA di crittografia a chiave pubblica**. Ci concentreremo in particolare su come generare:

- la **chiave pubblica** che qualunque mittente può usare per crittografare testo in chiaro in testo cifrato per un particolare destinatario;
- la **chiave privata** che solo quel particolare destinatario può usare per decrittografare il testo cifrato.

I fondamenti matematici dell'RSA sono sofisticati, mentre sono semplici i passi necessari per generare le chiavi pubblica e privata, per crittografare i messaggi con la chiave pubblica e per decrittografarli con la chiave privata, come vedremo a breve. A livello industriale l'RSA opera con numeri primi enormi, costituiti da centinaia di cifre. Nelle nostre spiegazioni useremo solamente numeri primi piccoli, per semplicità e per consentirvi di costruire velocemente una versione funzionante in scala ridotta dell'RSA. Le versioni dell'RSA che usano numeri primi piccoli non sono molto sicure, ma vi serviranno a capirne il funzionamento.

Crittografia a chiave pubblica

Whitfield Diffie e Martin Hellman, nella loro pubblicazione "New Directions in Cryptography",¹⁹ hanno introdotto la **crittografia a chiave pubblica** come risposta al problema della crittografia a chiave segreta, ovvero la vulnerabilità della chiave segreta, che deve essere nota sia al mittente sia al destinatario. Essi hanno avuto l'idea di questo schema, ma non hanno elaborato la sua implementazione.

Crittografia a chiave pubblica RSA

Rivest, Shamir e Adelman furono i primi a pubblicare un'implementazione funzionante di crittografia a chiave pubblica. Lo schema è chiamato RSA²⁰, dalle iniziali dei loro cognomi, ed è uno degli schemi di crittografia a chiave pubblica tra i più diffusi al mondo.²¹ Poiché questo schema può essere lento,²² molte organizzazioni preferiscono continuare a usare la crittografia a chiave privata, più veloce, servendosi invece della crittografia RSA per inviare in maniera sicura la chiave segreta.

Cenni storici

Nel Regno Unito, molti anni prima della pubblicazione dell'articolo sull'RSA,²³ Clifford Cocks ideò uno schema a chiave pubblica funzionante, ma dato che il suo lavoro era considerato confidenziale, non venne rivelato se non circa 20 anni dopo la comparsa dell'RSA.

La società RSA Security era in possesso di un brevetto sull'algoritmo RSA. Nel 2000, quando avrebbe dovuto rinnovare il brevetto, la società decise invece di rendere l'algoritmo di dominio pubblico.²⁴

Fasi dell'algoritmo RSA

Le *Fasi 1-5* qui sotto usano valori interi piccoli per spiegare il modo in cui l'algoritmo RSA genera una coppia chiave pubblica/chiave privata. In seguito, la *Fase 6* usa la chiave pubblica per crittografare testo in chiaro in testo cifrato, e la *Fase 7* usa la chiave privata per decrittografare il testo cifrato e restituire il testo originale in chiaro. Le fasi presentate sono basate sulla pubblicazione originale degli ideatori dell'RSA²⁵ e sulla pagina di Wikipedia relativa all'algoritmo RSA.²⁶

18. "New Directions in Cryptography." Accesso 8 gennaio 2021. <https://ee.stanford.edu/~hellman/publications/24.pdf>.
19. "New Directions in Cryptography." Accesso 8 gennaio 2021. <https://ee.stanford.edu/~hellman/publications/24.pdf>.
20. R. Rivest; A. Shamir; L. Adleman (febbraio 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" (PDF). Communications of the ACM. 21 (2): 120-126. <https://people.csail.mit.edu/rivest/Rsapaper.pdf>.
21. "RSA algorithm (Rivest-Shamir-Adleman)." Accesso 8 gennaio 2021. <https://searchsecurity.techtarget.com/definition/RSA>.
22. "RSA (cryptosystem)." Accesso 6 gennaio 2021. [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem)).
23. "Clifford Cocks." Accesso 8 gennaio 2021. https://en.wikipedia.org/wiki/Clifford_Cocks.
24. "RSA Security Releases RSA Encryption Algorithm into Public Domain." Accesso 8 gennaio 2021. https://web.archive.org/web/20071120112281/http://www.rsa.com/press_release.aspx?id=261.
25. Rivest, R.L., A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" (1977). Accesso 8 gennaio 2021. <https://people.csail.mit.edu/rivest/Rsapaper.pdf>.
26. "RSA Algorithm." Accesso 6 gennaio 2021. https://simple.wikipedia.org/wiki/RSA_algorithm.

Fase 1 dell'algoritmo RSA: scegliere due numeri primi

Selezionate due differenti numeri primi p e q . In questo caso pratico, utilizzeremo numeri primi piccoli: $p = 13$ e $q = 17$. Così sarà più facile gestire i calcoli nelle nostre analisi e sul computer usando il C, che per gli interi ha tipi di dati predefiniti e con un intervallo limitato. Nei sistemi di crittografia RSA di livello commerciale, questi numeri primi sono scelti casualmente e hanno centinaia di cifre. Per avere un'idea di quanto possano essere grandi gli interi in un algoritmo RSA, consultate la pagina web

https://en.wikipedia.org/wiki/RSA_numbers

che mostra diversi numeri interi da 100 a 617 cifre. I tipi di dati interi del C int, long int e long long int non possono contenere interi di tali dimensioni, quindi sono necessarie elaborazioni particolari per lavorare con numeri così grandi.

Fase 2 dell'algoritmo RSA: calcolare il modulo (n), che è parte delle chiavi pubblica e privata

Calcolate il modulo n , che è semplicemente il prodotto di p e q :

$$n = p * q$$

Dato che $p = 13$ e $q = 17$, n è 221. Come vedrete, n è parte sia della chiave pubblica che di quella privata. I valori di p e di q rimangono privati.

Fase 3 dell'algoritmo RSA: calcolare la funzione toziente

Calcolate $\Phi(n)$, che è la funzione toziente di Eulero.²⁷ Si calcola come:

$$\Phi(n) = (p - 1) * (q - 1)$$

Dati $p = 13$ e $q = 17$, $\Phi(n)$ è

$$\Phi(n) = 12 * 16 = 192$$

Questo numero viene usato nei calcoli che determinano l'esponente di crittografia (e) e l'esponente di decrittografia (d), che ci serviranno rispettivamente per crittografare testo in chiaro e decrittografare testo cifrato, come vedremo qui di seguito.

Fase 4 dell'algoritmo RSA: selezionare l'esponente della chiave pubblica (e) per i calcoli di crittografia

Ora dobbiamo scegliere un esponente, e , per la crittografia, soggetto alle regole seguenti:

- $1 < e < \Phi(n)$;
- e deve essere coprimo con $\Phi(n)$.

Due numeri interi si dicono coprimi se non hanno fattori primi in comune diversi da 1.

Nel nostro esempio, gli interi che soddisfano la prima regola, dato $\Phi(n) = 192$, sono i valori da 2 a 191. La scomposizione in fattori primi di 192 è

$$192 = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 3$$

Il valore di e deve essere coprimo con $\Phi(n)$, quindi non possiamo prendere in considerazione né i fattori primi né tutti i loro multipli. Pertanto vanno eliminati il valore 2 e tutti i numeri pari tra 2 e 190, così come il valore 3 e tutti i suoi multipli. Rimangono quindi come valori possibili per e :

5	7	11	13	17	19	23	25	29	31	35	37	41	43	47	49
53	55	59	61	65	67	71	73	77	79	83	85	89	91	95	97
101	103	107	109	113	115	119	121	125	127	131	133	137	139	143	145
149	151	155	157	161	163	167	169	173	175	179	181	185	187	191	

Ognuno di questi valori può essere usato come esponente per la chiave pubblica (e) per la crittografia. Per il nostro esempio, scegliamo il valore 37, quindi la nostra chiave pubblica è (37, 221).

27. "Euler's totient function." Accesso 7 gennaio 2021. https://en.wikipedia.org/wiki/Euler%27s_totient_function.

Fase 5 dell'algoritmo RSA: selezionare l'esponente della chiave privata (d) per i calcoli di decrittografia

La fase finale consiste nel determinare l'esponente della chiave privata, d, per la decrittografia. Il valore scelto per d deve essere tale che

$$(d * e) \bmod \Phi(n) = 1$$

Nel nostro esempio, il primo valore di d che rispetta questa regola è 109. Possiamo verificare se il calcolo precedente ha come risultato 1 inserendo i valori di d, e e $\Phi(n)$:

$$(109 * 37) \bmod 192$$

Il valore di $109 * 37$ è 4033. Moltiplicando 192 per 21, si ottiene 4032, con il resto di 1. Perciò, 109 è un valore valido per d. Ci sono molti valori potenziali per d: ciascuno di essi corrisponde a 109 più un multiplo del toziente (192). Per esempio, 301 ($109 + 1 * 192$):

$$(301 * 37) \bmod 192$$

$301 * 37$ dà 11137, che ha un resto di 1 se diviso per 192 ($192 * 58$ dà 11136, lasciando un resto di 1). Tutti i valori come quelli seguenti potrebbero essere usati per l'esponente d:

$$109 \quad 301 \quad 493 \quad 685 \quad 877 \quad \dots$$

Noi abbiamo scelto 109, quindi la nostra chiave privata è (109, 221).

Crittografare un messaggio con l'RSA

Una volta ottenuta la chiave pubblica, è facile crittografare un messaggio usando l'RSA. **Dato un messaggio in chiaro in forma di numero intero (M) da crittografare in un messaggio cifrato (C) e una chiave pubblica costituita da due numeri interi positivi e (per la crittografia) e n – rappresentati solitamente come (e, n) – un mittente può crittografare M con il calcolo seguente:**

$$C = M^e \bmod n$$

Il valore di M deve essere compreso nell'intervallo $0 \leq M < n$. Altrimenti, dovete spezzare il messaggio in più valori che siano compresi in questo intervallo e crittografare ciascuno separatamente.

Crittografiamo ora il valore 122 di M usando la nostra chiave pubblica (37, 221):

$$C = 122^{37} \bmod 221$$

Il valore 122^{37} è un numero enorme, ma si può eseguire questo calcolo con Wolfram Alpha sul sito

<https://www.wolframalpha.com/input/>

Inserite il calcolo come mostrato di seguito (il simbolo ^ in Wolfram Alpha rappresenta l'elevamento a potenza):

$$122^{\wedge}37 \bmod 221$$

Il risultato sarà 5, il nostro testo cifrato.

Decrittografare un messaggio con l'RSA

È facile anche decrittografare un messaggio anche quando si è in possesso della chiave privata. **Dato un messaggio cifrato in forma di numero intero (C) da decrittografare nel messaggio originale in chiaro (M) e una chiave privata costituita da due numeri interi positivi d e n – rappresentati solitamente come (d, n) – un destinatario del messaggio può decrittografare C con il calcolo seguente:**

$$M = C^d \bmod n$$

Decrittografiamo ora il valore 5 di C usando la nostra chiave privata (109, 221):

$$C = 5^{109} \bmod 221$$

Ancora, il valore 5^{109} è un numero enorme, ma si può eseguire questo calcolo con Wolfram Alpha inserendolo come segue:

$5^{109} \bmod 221$

Il risultato sarà 122, il nostro testo in chiaro.

Osservate che n è parte di *entrambe* le chiavi, quella pubblica e quella privata, e che il valore dell'esponente d si basa sull'esponente e e sul valore del modulo n .

Crittografia e decrittografia di stringhe

Supponiamo che vogliate usare l'algoritmo RSA per crittografare un messaggio di testo, come

Damn the torpedoes, full speed ahead!²⁸

Come sapete, l'algoritmo RSA può crittografare solo messaggi costituiti da numeri interi nell'intervallo $0 \leq M < n$. Quindi, per crittografare il messaggio precedente, dovete mappare i caratteri a valori interi.

Un modo per convertire i caratteri in numeri interi è quello di utilizzare il valore numerico di ciascun carattere nell'insieme di caratteri sottostante. Per questo esercizio, assumete i caratteri ASCII, che hanno valori interi nell'intervallo 0-127 (vedi Appendice B). Purché il valore intero di un carattere sia minore di n , potete crittografare quel valore come mostrato in precedenza e memorizzare ogni intero del testo cifrato risultante in un array di interi. Se cercate di stampare questi interi "cifrati" come caratteri, potrete ottenere strani simboli. Per esempio, gli interi del testo cifrato potrebbero rappresentare caratteri speciali, come newline o tabulazioni, oppure trovarsi al di fuori dell'intervallo del set ASCII. Per decriptografare poi il testo cifrato, prendete ogni numero intero risultante, convertitelo in un tipo `char` e memorizzatelo in un array di caratteri che rappresenterà il testo in chiaro decifrato. Assicuratevi di aver inserito il carattere nullo di terminazione nella stringa prima di stamparla.

Programmazione dell'algoritmo RSA

Implementate ora l'algoritmo RSA in C. Fate in modo che l'utente possa crittografare e decriptografare prima un semplice intero, poi una riga di testo. Il vostro programma dovrebbe produrre una finestra di dialogo di output simile alla seguente.

```
Enter a prime number for p: 13
Enter a prime number for q: 17
n is 221
totient is 192
Candidates for e: 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 53 55 59 61 65 67
71 73 77 79 83 85 89 91 95 97 101 103 107 109 113 115 119 121 125 127 131 133 137
139 143 145 149 151 155 157 161 163 167 169 173 175 179 181 185 187 191
Select a value for e from the preceding candidates: 37
Candidate for d: 109
Select a value for d--either the d candidate above
or d plus a multiple of the totient: 109
Enter a non-negative integer less than n to encrypt: 122
```

28. David Glasgow Farragut: ufficiale dell'Unione nella guerra di secessione americana e primo ammiraglio della marina militare degli Stati Uniti. Accesso 8 gennaio 2021. https://en.wikipedia.org/wiki/David_Farragut.

```
The ciphertext is: 5
The decrypted plaintext is: 122
Enter a sentence to encrypt:
Damn the torpedoes, full speed ahead!
The ciphertext is:
DG'
ue
;X}eW;es9 fh s}eeW GueGW!
The decrypted plaintext is:
Damn, the torpedoes,, full speed ahead!
```

Durante l'implementazione dell'algoritmo RSA, tenete presente i suggerimenti seguenti.

- **Elevamento a potenza modulare:** elevare un messaggio di testo in chiaro a un esponente grande (es. 122^{37}) dà come risultato valori enormi che i tipi di dati predefiniti in C per gli interi, che hanno un intervallo limitato, non sono in grado di rappresentare. Come sapete, i calcoli di crittografia e decrittografia eseguono operazioni sia di elevamento a potenza che di modulo. Si possono combinare queste operazioni tramite l'elevamento a potenza modulare per mantenere i calcoli di crittografia e decrittografia dell'RSA entro intervalli ragionevoli. Definite una funzione chiamata `modularPow` che esegue l'elevamento a potenza modulare. Per l'algoritmo di elevamento a potenza modulare, trovate lo pseudocodice all'indirizzo:

https://en.wikipedia.org/wiki/Modular_exponentiation#Memory-efficient_method
- **Calcolo del massimo comun divisore.** I valori candidati per l'esponente e (*Fase 4*) devono essere coprimi con il toziente; anche in questo caso, l'unico fattore comune è 1. Per determinare se due numeri sono coprimi, vi serve una **funzione `gcd`** che calcola il **massimo comun divisore di due interi**. Il vostro programma deve visualizzare tutti i possibili valori candidati per e . L'Esercizio 5.29 vi ha richiesto di scrivere una funzione `gcd`.
- **Verifica dei numeri primi.** L'algoritmo RSA richiede due numeri primi, p e q . Dovete definire una **funzione `isPrime`** che **determina se un intero è effettivamente un numero primo**: usate questa funzione per confermare che i valori p e q inseriti dall'utente siano numeri primi. L'Esercizio 6.30 vi ha richiesto di implementare il Setaccio di Eratostene per trovare i numeri primi.

Il vostro programma deve inoltre definire le funzioni seguenti:

- **una funzione per crittografare un messaggio in chiaro M usando la chiave pubblica (e , n):**

```
int encrypt(int M, int e, int n);
```
- **una funzione per decriptografare un messaggio cifrato C usando la chiave privata (d , n):**

```
int decrypt(int C, int d, int n);
```
- **una funzione per crittografare una stringa chiamando la funzione `encrypt` per ogni carattere della stringa e inserendo i risultati in un array di interi:**

```
void encryptString(
    char* plaintext, int ciphertext[], int e, int n);
```
- **una funzione per decriptografare un testo cifrato, a partire da un array di interi, chiamando la funzione `decrypt` per ogni intero e inserendo i risultati in un array di caratteri.** La dimensione del parametro rappresenta il numero di caratteri che sono stati crittografati. Assicuratevi di terminare la stringa in `decryptedPlaintext` con un carattere nullo ('\0'):


```
char* decryptedPlaintext;
```

```
void decryptString(int ciphertext[],
    char decryptedPlaintext[], size_t size, int d, int n);
```

Riferimenti

La seguente presentazione video in due parti offre una chiara spiegazione dell'algoritmo RSA.

- The RSA Encryption Algorithm (1 of 2: Computing an Example):²⁹
<https://www.youtube.com/watch?v=4zahvcJ9g1g>
- The RSA Encryption Algorithm (2 of 2: Generating the Keys):³⁰
<https://www.youtube.com/watch?v=o0cTVTpUsPQ>

8.44 (Perfezionamento dell'algoritmo RSA) Nel 1998, l'algoritmo RSA è stato perfezionato sostituendo $\Phi(n)$ con $\lambda(n)$ (che si pronuncia "lambda di n"):^{31,32}

$$\lambda(n) = \text{lcm}((p - 1), (q - 1))$$

dove lcm rappresenta il **minimo comune multiplo** (*least common multiple*).³³ Abbiamo usato $\Phi(n)$ nell'esercizio precedente sull'RSA con $p = 13$ e $q = 17$. Il nuovo calcolo corrispondente con $\lambda(n)$ diventa

$$\lambda(n) = \text{lcm}(12, 16)$$

dove il minimo comune multiplo di 12 e 16 è 48, come si può vedere dalle seguenti liste di multipli:

12	24	36	48	...
16	32	48	60	...

Create una copia del codice che avete scritto come soluzione dell'esercizio precedente e sostituite ogni occorrenza di $\Phi(n)$ con $\lambda(n)$, poi testate il vostro codice aggiornato con gli stessi valori (numeri primi) usati per p e q . Quando crittografate il testo in chiaro con l'approccio $\lambda(n)$, il testo cifrato risultante sarà probabilmente differente, ma il testo in chiaro decrittografato dovrebbe essere il medesimo.

8.45 (Mettere alla prova i limiti del vostro algoritmo RSA) Mettete alla prova il vostro programma aumentando gradualmente i valori per p e q . Quanto possono crescere questi valori prima che il programma smetta di funzionare? Testate inoltre il vostro programma con candidati sempre più alti per e e d .

8.46 (Miglioramenti al vostro codice RSA) Modificate il vostro programma RSA come segue.

- Il programma ha visualizzato tutti i possibili candidati per l'esponente di crittografia e . Modificate lo in modo che mostri i primi cinque potenziali valori per l'esponente di decrittografia d (ovvero, il primo valore di d più $1 * \text{totient}$, il primo valore di d più $2 * \text{totient}$, ecc.). Fate seguire la vostra lista di possibilità da un'ellissi (...).
- Via via che diventeranno più grandi, alla fine i vostri numeri primi p e q arriveranno a superare il limite massimo di valore per il tipo `int`, che potete trovare in `<limits.h>`. Modificate il vostro codice in modo che esegua tutti i calcoli RSA con gli interi usando il tipo `long long int`. Tenete presente che nemmeno questo tipo è adeguato per contenere gli interi enormi che si usano nell'RSA di livello industriale. È necessaria una programmazione speciale per utilizzare valori interi di tali dimensioni. Ricordate di modificare in `%lld` tutti gli specificatori di conversione `%d` delle istruzioni `printf` e `scanf`.

29. Eddie Woo (misterwootube). "The RSA Encryption Algorithm (1 of 2: Computing an Example)," 4 novembre 2014. <https://www.youtube.com/watch?v=4zahvcJ9g1g>.

30. Eddie Woo (misterwootube). "The RSA Encryption Algorithm (2 of 2: Generating the Keys)," 4 novembre 2014. <https://www.youtube.com/watch?v=o0cTVTpUsPQ>.

31. "PKCS #1: RSA Cryptography Specifications, Version 2.0." Accesso 7 gennaio 2021. <https://tools.ietf.org/html/rfc2437>.

32. "RSA Algorithm." Accesso 7 gennaio 2021. https://simple.wikipedia.org/wiki/RSA_algorithm.

33. "Least common multiple." Accesso 7 gennaio 2021. https://en.wikipedia.org/wiki/Least_common_multiple.

8.47 (Progetto impegnativo: il Problema RSA³⁴) In questo esercizio, farete ricerche sugli attacchi che sono stati compiuti contro implementazioni RSA di livello industriale. Poi proverete a “rompere” un testo cifrato creato dall’implementazione RSA in scala ridotta che avete costruito nell’Esercizio 8.43. Come già detto, questo tipo di implementazioni in scala ridotta non sono sicure.

- a) Ricercate il genere di attacchi che sono stati compiuti contro sistemi RSA di livello industriale. Osservate quali tipi di attacchi hanno avuto successo e quali invece hanno fallito.
- b) La forza dell’RSA deriva dagli enormi numeri primi p e q (che di solito hanno centinaia di cifre ciascuno) usati per calcolare il valore ancora più enorme di n (che è $p * q$) e dal carico computazionale necessario per scomporre n in fattori primi e trovare p e q . Il “Problema RSA” consiste nel riuscire a decrittografare un testo cifrato possedendo solo la chiave pubblica (e , n). Per svolgere questo compito è necessario trovare i fattori primi di n , p e q , dai quali si può quindi derivare d e decrittografare il testo cifrato.

Supponete di avere una chiave pubblica (e , n) e un testo cifrato che è stato crittografato usando quella chiave con la vostra implementazione RSA in scala ridotta, ma di non essere a conoscenza della chiave privata richiesta per decrittografare il testo cifrato. Utilizzate le tecniche di calcolo a forza bruta per trovare i fattori primi di n , p e q . Eseguite poi i calcoli necessari per recuperare d e decrittografare il messaggio.

34. “RSA Problem.” Accesso 8 gennaio 2021. https://en.wikipedia.org/wiki/RSA_problem.

CAPITOLO

9

Sommario del capitolo

- 9.1 Introduzione
- 9.2 Stream
- 9.3 Formattazione dell'output con `printf`
- 9.4 Stampa di interi
- 9.5 Stampa di numeri in virgola mobile
- 9.6 Stampa di stringhe e caratteri
- 9.7 Altri specificatori di conversione
- 9.8 Stampare con larghezza di campo e precisione
- 9.9 Flag di formato di `printf`
- 9.10 Stampa di letterali e sequenze di escape
- 9.11 Input formattato con `scanf`
- 9.12 Programmazione sicura in C
- 9.13 Riepilogo

Input/output formattato

Obiettivi

- Usare stream di input e output.
- Usare le funzionalità per la formattazione nelle operazioni di stampa.
- Usare le funzionalità per la formattazione nelle operazioni di input.
- Stampare interi, numeri in virgola mobile, stringhe e caratteri.
- Stampare specificando la larghezza di campo e la precisione.
- Usare dei flag nella stringa di controllo del formato con `printf`.
- Stampare letterali e sequenze di escape.
- Leggere input formattato usando `scanf`.

9.1 Introduzione

Una parte importante della soluzione di un problema è la presentazione dei risultati. In questo capitolo approfondiremo le caratteristiche di formattazione di `printf` e `scanf`, che rispettivamente inviano in uscita dati allo stream standard output e leggono dati dallo stream standard input. Includete l'intestazione `<stdio.h>` nei programmi che chiamano queste funzioni. Il Capitolo 11 esaminerà diverse ulteriori funzioni incluse nella libreria standard di input/output (`<stdio.h>`).

9.2 Stream

Tutte le operazioni di input e output vengono eseguite con sequenze di byte chiamate **stream**:

- nelle operazioni di input i byte fluiscono da un dispositivo (es. una tastiera, un disco rigido, una connessione di rete) alla memoria principale;
- nelle operazioni di output i byte fluiscono dalla memoria principale a un dispositivo (es. uno schermo, una stampante, un disco rigido, una connessione di rete, ecc.).

Quando inizia l'esecuzione, il programma ha accesso a tre stream:

- lo stream standard input, che è connesso alla tastiera;
- lo stream standard output, che è connesso allo schermo;
- lo stream standard error, anch'esso collegato allo schermo.

I sistemi operativi permettono spesso che questi stream siano reindirizzati ad altri dispositivi. Nel Capitolo 11 gli stream saranno esaminati in dettaglio.

✓ Autovalutazione

- (Completere)* Potete _____ gli stream standard ad altri dispositivi.

Risposta: reindirizzare.

- (Scelta multipla)* Quale delle seguenti affermazioni è *falsa*?

- a) Le operazioni di input e output vengono eseguite con array, che sono sequenze di byte.
- b) Nelle operazioni di input, i byte fluiscono da un dispositivo alla memoria principale.
- c) Nelle operazioni di output, i byte fluiscono dalla memoria principale a un dispositivo.
- d) Quando inizia l'esecuzione, gli stream standard sono collegati al programma.

Risposta: a) è *falsa*. In realtà, le operazioni di input e output vengono eseguite con stream, che sono sequenze di byte.

9.3 Formattazione dell'output con printf

Nel corso del libro, avete visto diverse funzionalità per la formattazione dell'output con `printf`. Ogni chiamata di `printf` contiene una **stringa di controllo del formato** che descrive il formato dell'output. La stringa di controllo del formato è costituita da **specificatori di conversione**, **flag** (indicatori), **larghezze di campo**, **precisioni** e **caratteri letterali**. Insieme al segno di percentuale (%), questi formano le **specifiche di conversione**. La funzione `printf` può effettuare le seguenti funzionalità di formattazione:

- Arrotondare** i valori in virgola mobile a un numero indicato di cifre decimali.
- Allineare colonne di numeri al loro punto decimale.
- Allineare a destra** e **allineare a sinistra** gli output.
- Inserire caratteri letterali in posizioni precise in una riga di output.
- Rappresentare numeri in virgola mobile in formato esponenziale.
- Rappresentare interi senza segno in formato ottale ed esadecimale. Consultate l'Appendice E per maggiori informazioni sui valori ottali ed esadecimali.
- Stampare tutti i tipi di dati con precisione e larghezza di campo fissata.

La funzione `printf` ha la forma

`printf(stringa di controllo del formato, altri argomenti);`

La *stringa di controllo del formato* descrive il formato dell'output e gli *altri argomenti* (che sono facoltativi) corrispondono a ogni specifica di conversione nella *stringa di controllo del formato*. Ogni specifica di conversione inizia con un segno di percentuale (%) e termina con uno specificatore di conversione. Vi possono essere molte specifiche di conversione in una stringa di controllo del formato.

✓ Autovalutazione

- (Completere)* Ogni chiamata di `printf` contiene una _____ che descrive il formato dell'output.

Risposta: stringa di controllo del formato.

- (Scelta multipla)* Quale delle seguenti è una funzionalità di formattazione che la funzione `printf` può eseguire?

- a) Arrotondamento di valori in virgola mobile a un numero indicato di cifre decimali, e allineamento di una colonna di numeri al loro punto decimale.
- b) Rappresentazione di numeri in virgola mobile in formato esponenziale. Rappresentazione di interi senza segno in formato ottale ed esadecimale.

- c) Stampa di tutti i tipi di dati con precisione e larghezza di campo fissata.
 d) Tutte le precedenti sono funzionalità di formattazione di `printf`.

Risposta: d.

9.4 Stampa di interi

Un intero è un numero senza punto decimale, come 776, 0 o -52. I valori interi possono essere stampati in uno dei diversi formati descritti dai seguenti **specificatori di conversione di interi**.

Specificatore di conversione	Descrizione
d	Stampa come un intero decimale con segno.
i	Stampa come un intero decimale con segno.
o	Stampa come un intero ottale senza segno.
u	Stampa come un intero decimale senza segno.
x o X	Stampa come un intero esadecimale senza segno. X usa le cifre 0-9 e le lettere maiuscole A-F, e x usa le cifre 0-9 e le lettere minuscole a-f.
-h, l o ll (lettera "elle")	Questi modificatori di lunghezza vanno posti prima di uno specificatore di conversione di interi per indicare che il valore da stampare è di tipo intero short, long o long long.

Il programma della Figura 9.1 stampa un intero usando ognuno degli specificatori di conversione di interi. Come impostazione predefinita, il segno più non viene stampato, ma vedremo in seguito come fare in modo che i segni più vengano stampati. Le righe 10 e 11 usano gli specificatori di conversione hd e ld per stampare valori interi short e long. Il suffisso L sul letterale 2000000000L indica che il suo tipo è long; C tratta i letterali di numeri interi come int. Stampare un valore negativo con uno specificatore di conversione che si aspetta un valore unsigned è un errore logico. Quando la riga 14 stampa -455 con %u, il risultato è il valore unsigned 4294966841. Un piccolo valore negativo viene visualizzato come un numero intero positivo grande a causa del "bit del segno" del valore nella rappresentazione binaria sottostante. Consultate l'Appendice E (disponibile sulla piattaforma MyLab) per una discussione sul sistema numerico binario e sul bit del segno.

```

1 // fig09_01.c
2 // Uso di specificatori di conversione di interi
3 #include <stdio.h>
4
5 int main(void) {
6     printf("%d\n", 455);
7     printf("%i\n", 455); // i come d in printf
8     printf("%d\n", +455); // non viene stampato il segno più
9     printf("%d\n", -455); // viene stampato il segno meno
10    printf("%hd\n", 32000); // stampa come tipo short
11    printf("%ld\n", 2000000000L); // stampa come tipo long
12    printf("%o\n", 455); // ottale
13    printf("%u\n", 455);
14    printf("%x\n", 455); // esadecimale con lettere minuscole
15    printf("%X\n", 455); // esadecimale con lettere maiuscole
16
17 }
```

```

455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7

```

Figura 9.1 Uso di specificatori di conversione di interi.

✓ Autovalutazione

- (Scelta multipla) Quale specificatore di conversione di interi è descritto da "Stampa come un intero decimale senza segno"?
 - ud.
 - ui.
 - u.
 - Nessuno dei precedenti.

Risposta: c.

- (Cosa fa questo codice?) Mostrate esattamente cosa stampa il seguente codice:

```

printf("%d\n", 235);
printf("%i\n", 235);
printf("%d\n", +235);
printf("%d\n", -235);

```

Risposta:

```

235
235
235
-235

```

9.5 Stampa di numeri in virgola mobile

I valori in virgola mobile contengono un punto decimale, come in 33.5, 0.0 o -657.983. I valori in virgola mobile sono stampati usando gli specificatori di conversione riassunti di seguito.

Specificatore di conversione	Descrizione
e o E	Stampa un valore in virgola mobile in notazione esponenziale.
f o F	Stampa i valori in virgola mobile nella notazione in virgola fissa.
g o G	Stampa un valore in virgola mobile o nel formato f oppure nel formato esponenziale e (o E), in base alla grandezza del valore.
L	Questo modificatore di lunghezza va posto prima dello specificatore di conversione di numeri in virgola mobile per indicare che viene stampato un valore in virgola mobile long double.

Notazione esponenziale

Gli specificatori di conversione e e E stampano valori in virgola mobile in **notazione esponenziale**, l'equivalente informatica della **notazione scientifica** utilizzata in matematica. Per esempio, il valore 150.4582 è rappresentato in notazione scientifica come

1.504582 × 10²

e in notazione esponenziale come

1.504582E+02

In questa notazione, la E sta per “esponente” e indica che 1.504582 è moltiplicato per 10 elevato alla seconda (E+02).

9.5.1 Specificatori di conversione e, E e f

I valori stampati con gli specificatori di conversione e, E e f mostrano per impostazione predefinita sei cifre di precisione alla destra del punto decimale (es., 1.045927); altre precisioni si possono specificare esplicitamente.

Lo specificatore di conversione f stampa sempre almeno una cifra alla sinistra del punto decimale, quindi i valori frazionari saranno preceduti da "0.". Gli specificatori di conversione e ed E precedono l'esponente con la e minuscola o con la E maiuscola, e stampano esattamente una sola cifra alla sinistra del punto decimale.

9.5.2 Specificatori di conversione g e G

Lo specificatore di conversione g (o G) stampa sia nel formato e (E) che nel formato f senza zeri finali, quindi 1.234000 è stampato come 1.234. Lo specificatore di conversione g utilizza il formato e (E) se, dopo la conversione nella notazione esponenziale, l'esponente del valore è minore di -4 o maggiore o uguale alla precisione specificata. Altrimenti, g utilizza lo specificatore di conversione f per stampare il valore. La precisione predefinita è di sei cifre significative per g e G, cioè verrà stampato un massimo di sei cifre.

È richiesta almeno una cifra decimale perché sia stampato il punto decimale. Per esempio, i valori 0.0000875, 8750000.0, 8.75 e 87.50 sono stampati come 8.75e-05, 8.75e+06, 8.75 e 87.5 con lo specificatore di conversione g. Il valore 0.0000875 usa la notazione e perché, quando è convertito nella notazione esponenziale, il suo esponente (-5) è minore di -4. Il valore 8750000.0 usa la notazione e perché il suo esponente (6) è uguale alla precisione predefinita.

Precisione

La precisione per gli specificatori di conversione g e G indica il numero massimo di cifre significative da stampare, compresa la cifra alla sinistra del punto decimale. Quindi, il valore 1234567.0 viene stampato come 1.23457e+06, usando la specifica di conversione %g. Ricordate che tutti gli specificatori di conversione di numeri in virgola mobile hanno una precisione predefinita di 6. Vi sono sei cifre significative nel risultato: 1 a sinistra del punto decimale e 23457 a destra. Per la notazione esponenziale, g e G precedono l'esponente con una e minuscola o una E maiuscola. Quando stampate dati, assicuratevi che l'utente sia informato delle situazioni in cui i dati possono essere imprecisi a causa della formattazione (es. errori di arrotondamento dovuti alla precisione specificata).

9.5.3 Dimostrare gli specificatori di conversione in virgola mobile

Il programma della Figura 9.2 illustra l'uso di ognuno degli specificatori di conversione di numeri in virgola mobile. Le specifiche di conversione %E, %e e %g fanno sì che il valore sia arrotondato, mentre lo specificatore di conversione %f non lo fa.

```

1 // fig09_02.c
2 // Uso degli specificatori di conversione di numeri in virgola mobile
3 #include <stdio.h>
4
5 int main(void) {
6     printf("%e\n", 1234567.89);

```

```

7   printf("%e\n", +1234567.89); // non viene stampato il segno piu'
8   printf("%e\n", -1234567.89); // viene stampato il segno meno
9   printf("%E\n", 1234567.89);
10  printf("%f\n", 1234567.89); // sei cifre a destra del punto decimale
11  printf("%g\n", 1234567.89); // stampa con la lettera minuscola e
12  printf("%G\n", 1234567.89); // stampa con la lettera maiuscola E
13 }

```

```

1.234568e+06
1.234568e+06
-1.234568e+06
1.234568E+06
1234567.890000
1.23457e+06
1.23457E+06

```

Figura 9.2 Uso degli specificatori di conversione di numeri in virgola mobile.

✓ Autovalutazione

1. (*Completare*) Gli specificatori di conversione e ed E stampano valori in notazione esponenziale, l'equivalente informatica della _____ utilizzata in matematica.

Risposta: notazione scientifica.

2. (*Scelta multipla*) Quale affermazione sugli specificatori di conversione e, E e f è *falsa*?

- I valori stampati con gli specificatori di conversione e, E e f mostrano sei cifre di precisione a destra del punto decimale come impostazione predefinita.
- Lo specificatore di conversione f stampa sempre esattamente una cifra a sinistra del punto decimale.
- Gli specificatori di conversione e ed E stampano rispettivamente la e minuscola e la E maiuscola, precedendo l'esponente, e stampano esattamente una sola cifra alla sinistra del punto decimale.
- Tutte le affermazioni precedenti sono *vere*.

Risposta: b) è *falsa*. In realtà, lo specificatore di conversione f stampa almeno una cifra alla sinistra del punto decimale.

9.6 Stampa di stringhe e caratteri

Gli specificatori di conversione c e s sono usati per stampare, rispettivamente, singoli caratteri e stringhe. Lo **specificatore di conversione c** richiede un argomento char. Lo **specificatore di conversione s** richiede un puntatore a char come argomento e fa sì che i caratteri siano stampati finché non si incontra un carattere nullo di terminazione ('\0'). Se la stringa non ha un terminatore nullo, il risultato è indefinito: printf continuerà a stampare finché infine non incontrerà un byte zero o il programma terminerà prematuramente (ovvero si arresterà) indicando un “errore di segmentazione” o una “violazione di accesso”. Il programma della Figura 9.3 stampa caratteri e stringhe con gli specificatori di conversione c e s.

```

1 // fig09_03.c
2 // Uso degli specificatori di conversione di caratteri e stringhe
3 #include <stdio.h>
4
5 int main(void) {
6     char character = 'A'; // inizializza un char
7     printf("%c\n", character);
8
9     printf("%s\n", "This is a string");

```

```

10
11     char string[] = "This is a string"; // inizializza un array di char
12     printf("%s\n", string);
13
14     const char *stringPtr = "This is also a string"; // puntatore a char
15     printf("%s\n", stringPtr);
16 }
```

A
 This is a string
 This is a string
 This is also a string

Figura 9.3 Uso degli specificatori di conversione di caratteri e stringhe.

Errori nelle stringhe di controllo del formato

La maggior parte dei compilatori non rileva errori nella stringa di controllo del formato. Solitamente non ci si accorge di tali errori finché un programma non fallisce o produce risultati scorretti al momento dell'esecuzione.

- ☒ ■ Usare %c per stampare una stringa è un errore logico; %c si aspetta un argomento char. Una stringa è un puntatore a char (cioè un char *).
- ☒ ■ Usare %s per stampare un argomento char causa spesso un errore irreversibile in fase d'esecuzione, chiamato violazione di accesso. La specifica di conversione %s si aspetta un argomento di tipo puntatore a char, quindi tratta il valore numerico di char come un puntatore. Valori numerici così piccoli rappresentano spesso indirizzi di memoria limitati dal sistema operativo.

✓ Autovalutazione

1. (*Completere*) Lo specificatore di conversione s fa sì che i caratteri siano stampati finché non si incontra un _____.

Risposta: carattere nullo di terminazione ('\\0').

2. (*Vero/Falso*) I compilatori rilevano gli errori nella stringa di controllo del formato, quindi non si riscontreranno risultati errati in fase di esecuzione.

Risposta: *Falso*. In realtà, la maggior parte dei compilatori non rileva errori nella stringa di controllo del formato. Solitamente non ci si accorge di tali errori finché un programma non fallisce o produce risultati scorretti al momento dell'esecuzione.

9.7 Altri specificatori di conversione

Considerate gli specificatori di conversione p e %:

- p stampa un valore puntatore in un formato definito dall'implementazione;
- % stampa il carattere di percentuale.

Il %p del programma della Figura 9.4 stampa il valore di ptr e l'indirizzo di x in un formato definito dall'implementazione, solitamente usando la notazione esadecimale. Le variabili ptr e x hanno valori identici poiché la riga 7 assegna l'indirizzo di x a ptr. Gli indirizzi visualizzati sul vostro sistema varieranno. L'ultima istruzione printf usa %% per stampare il carattere %; è richiesto %% perché printf normalmente tratta % come l'inizio di una specifica di conversione. Cercare di stampare un carattere letterale di percentuale usando % invece di %% nella stringa di controllo del formato è un errore. Quando appare % in una stringa di controllo del formato, deve essere seguito da uno specificatore di conversione.

```

1 // fig09_04.c
2 // Uso degli specificatori di conversione p e %
3 #include <stdio.h>
4
5 int main(void) {
6     int x = 12345;
7     int *ptr = &x;
8
9     printf("The value of ptr is %p\n", ptr);
10    printf("The address of x is %p\n\n", &x);
11
12    printf("Printing a %% in a format control string\n");
13 }

```

```

The value of ptr is 0x7ffff6eb911c
The address of x is 0x7ffff6eb911c

Printing a % in a format control string

```

Figura 9.4 Uso degli specificatori di conversione p e %.

✓ Autovalutazione

1. (*Completere*) Un'istruzione `printf` usa _____ per stampare il carattere %.

Risposta: %%.

2. (*Vero/Falso*) Lo specificatore di conversione p stampa un indirizzo usando la notazione decimale.

Risposta: Falso. In realtà, lo specificatore di conversione p stampa un indirizzo in un formato definito dall'implementazione, solitamente usando la notazione esadecimale.

9.8 Stampare con larghezza di campo e precisione

La dimensione esatta di un campo in cui sono stampati i dati è specificata da una **larghezza di campo**. Se la larghezza di campo è maggiore rispetto ai dati che vengono stampati, questi vengono normalmente allineati a destra all'interno del campo. Un intero che rappresenta la larghezza di campo è inserito tra il segno di percentuale (%) e lo specificatore di conversione (es., %4d).

9.8.1 Larghezze di campo per interi

Il programma della Figura 9.5 stampa due gruppi di cinque numeri ciascuno, allineando a destra i numeri contenenti meno cifre della larghezza di campo. I valori più ampi del campo vengono comunque stampati per intero. Notate che il segno meno di un valore negativo usa una posizione di un carattere nella larghezza di campo. Le larghezze di campo possono essere usate con tutti gli specificatori di conversione. Non fornire una larghezza di campo sufficientemente grande per rappresentare un valore da stampare può provocare lo spostamento di altri dati stampati e produrre output che generano confusione. Siate consapevoli dei vostri dati!



```

1 // fig09_05.c
2 // Allineamento a destra di interi in un campo
3 #include <stdio.h>
4
5 int main(void) {
6     printf("%4d\n", 1);
7     printf("%4d\n", 12);
8     printf("%4d\n", 123);

```

```

9  printf("%4d\n", 1234);
10 printf("%4d\n\n", 12345);
11
12 printf("%4d\n", -1);
13 printf("%4d\n", -12);
14 printf("%4d\n", -123);
15 printf("%4d\n", -1234);
16 printf("%4d\n", -12345);
17 }

```

```

1
12
123
1234
12345

-1
-12
-123
-1234
-12345

```

Figura 9.5 Allineamento a destra di interi in un campo.

9.8.2 Precisione per interi, numeri in virgola mobile e stringhe

La funzione `printf` permette inoltre di specificare la precisione con cui i dati sono stampati. La precisione ha significati differenti per i diversi tipi di dati:

- Quando si usa con specificatori di conversione di interi, la precisione indica il numero minimo di cifre da stampare. Se il valore stampato contiene meno cifre della precisione specificata e il valore di precisione ha uno zero iniziale o un punto decimale, vengono inseriti degli zeri come prefissi per il valore stampato fino a raggiungere il numero totale di cifre richiesto dalla precisione. Se nel valore di precisione non è presente né uno zero né un punto decimale, sono invece inseriti degli spazi. La precisione predefinita per gli interi è 1.
- Quando viene usata con gli specificatori di conversione di numeri in virgola mobile `e`, `E` e `f`, la precisione è il numero di cifre che compaiono dopo il punto decimale.
- Quando viene usata con gli specificatori di conversione `g` e `G`, la precisione è il numero massimo di cifre significative da stampare.
- Quando è usata con lo specificatore di conversione `s`, la precisione è il numero massimo di caratteri scritti dall'inizio della stringa.

Per specificare la precisione, mettete un punto decimale (.), seguito da un intero che rappresenta la precisione tra il segno di percentuale e lo specificatore di conversione. Il programma della Figura 9.6 illustra l'uso della precisione nelle stringhe di controllo del formato. Quando il valore in virgola mobile è stampato con una precisione più piccola del numero originario di cifre decimali del valore, questo è arrotondato.

```

1 // fig09_06.c
2 // Stampa di interi, numeri in virgola mobile e stringhe con precisioni
3 #include <stdio.h>
4
5 int main(void) {
6     puts("Using precision for integers");
7     int i = 873; // inizializza int i

```

```

8     printf("\t%.4d\n\t%.9d\n\n", i, i);
9
10    puts("Using precision for floating-point numbers");
11    double f = 123.94536; // inizializza double f
12    printf("\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f);
13
14    puts("Using precision for strings");
15    char s[] = "Happy Birthday"; // inizializza l'array di char s
16    printf("\t%.11s\n", s);
17 }

```

```

Using precision for integers
0873
000000873

Using precision for floating-point numbers
123.945
1.239e+02
124

Using precision for strings
Happy Birth

```

Figura 9.6 Stampa di interi, numeri in virgola mobile e stringhe con precisioni.

9.8.3 Combinare larghezze di campo e precisioni

La larghezza di campo e la precisione possono essere combinate inserendo la larghezza di campo, seguita da un punto decimale, seguito a sua volta da un valore di precisione tra il segno di percentuale e lo specificatore di conversione, come nell'istruzione

```
printf("%9.3f", 123.456789);
```

che stampa 123.457 con tre cifre alla destra del punto decimale allineato a destra in un campo di nove cifre.

Specificare larghezze di campo e precisioni come argomenti

È possibile specificare la larghezza di campo e la precisione usando espressioni intere nella lista degli argomenti che segue dopo la stringa di controllo del formato. Per usare questa modalità, inserite un asterisco (*) al posto della larghezza di campo o della precisione (o di entrambe). Il corrispondente argomento int nella lista degli argomenti viene valutato e usato al posto dell'asterisco. Un valore di larghezza di campo può essere positivo o negativo (il che fa sì che l'output sia allineato a sinistra nel campo, come descritto nel prossimo paragrafo). L'istruzione

```
printf("%*.*f", 7, 2, 98.736);
```

usa 7 per la larghezza di campo, 2 per la precisione e stampa il valore 98.74 allineato a destra.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è falsa?

- a) La precisione predefinita per gli interi è 1.
- b) Quando viene usata con gli specificatori di conversione di numeri in virgola mobile e, E e f, la precisione è il numero di cifre che compaiono prima del punto decimale.
- c) Quando viene usata con gli specificatori di conversione g e G, la precisione è il numero massimo di cifre significative da stampare.

- d) Quando viene usata con lo specificatore di conversione s, la precisione è il numero massimo di caratteri scritti dall'inizio della stringa.

Risposta: b) è falsa. Quando viene usata con gli specificatori di conversione di numeri in virgola mobile e, E e f, la precisione è il numero di cifre che compaiono dopo il punto decimale.

2. (*Cosa fa questo codice?*) Descrivete esattamente cosa stampa il seguente codice:

```
printf("%9.3f", 123.456789);
```

Risposta: Il codice allinea a destra in un campo di nove cifre il valore arrotondato 123.457.

3. (*Cosa fa questo codice?*) Descrivete esattamente cosa stampa il seguente codice:

```
printf("%*.2f", 7, 2, 98.736);
```

Risposta: L'istruzione usa 7 per la larghezza del campo, 2 per la precisione e stampa il valore 98.74 allineato a destra in un campo di 7.

9.9 Flag di formato di printf

La funzione printf permette di inserire anche dei *flag* per integrare le sue funzionalità di formattazione dell'output. Nella tabella seguente sono riepilogati i cinque flag che è possibile utilizzare nelle stringhe di controllo del formato.

Flag	Descrizione
- (segno meno)	Allinea a sinistra l'output dentro il campo specificato.
+	Stampa come prefisso il segno più per i valori positivi e il segno meno per i valori negativi.
spazio	Stampa uno spazio prima di un valore positivo non stampato con il flag +.
#	Antepone uno 0 al valore in output quando è usato con lo specificatore di conversione ottale o.
	Antepone 0x o 0X al valore in output quando è usato con lo specificatore di conversione esadecimale x o X.
	Forza la scrittura di un punto decimale per un numero in virgola mobile stampato con e, E, f, g o G che non contiene una parte frazionale. Normalmente il punto decimale è stampato solo se lo segue una cifra. Con gli specificatori g e G, gli zeri finali non vengono eliminati.
0 (zero)	Riempie un campo con zeri iniziali.

9.9.1 Allineamento a destra e a sinistra

I flag in una specifica di conversione vengono posizionati immediatamente a destra del % e prima dello specificatore di formato. È possibile combinare più flag in uno specificatore di conversione. Il programma della Figura 9.7 mostra l'allineamento a destra e a sinistra di una stringa, un intero, un carattere e un numero in virgola mobile. Le righe 6 e 8 stampano righe di numeri che rappresentano le posizioni delle colonne, dandovi la possibilità di confermare che l'allineamento a destra e a sinistra abbia funzionato correttamente.

```
1 // fig09_07.c
2 // Allineamento a destra e a sinistra di valori
3 #include <stdio.h>
4
5 int main(void) {
6     puts("1234567890123456789012345678901234567890");
7     printf("%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23);
```

```

8     puts("1234567890123456789012345678901234567890");
9     printf("%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23);
10    }

```

```

1234567890123456789012345678901234567890
hello      7      a  1.230000

1234567890123456789012345678901234567890
hello      7      a  1.230000

```

Figura 9.7 Allineamento a destra e a sinistra di valori.

9.9.2 Stampare numeri positivi e negativi con e senza il flag +

Il programma della Figura 9.8 stampa un numero positivo e un numero negativo, ognuno con e senza il flag +. Il segno meno è stampato in entrambi i casi, ma il segno più è stampato solo quando è usato il flag +.

```

1 // fig09_08.c
2 // Stampa di numeri positivi e negativi con e senza il flag +
3 #include <stdio.h>
4
5 int main(void) {
6     printf("%d\n%d\n", 786, -786);
7     printf("%+d\n%+d\n", 786, -786);
8 }

```

```

786
-786
+786
-786

```

Figura 9.8 Stampa di numeri positivi e negativi con e senza il flag +.

9.9.3 Usare il flag spazio

Il programma della Figura 9.9 antepone uno spazio a un numero positivo usando il flag spazio. Questo è utile per allineare i numeri positivi e negativi con lo stesso numero di cifre. Il valore -547 non è preceduto da uno spazio nell'output a causa del suo segno meno.

```

1 // fig09_09.c
2 // Uso del flag spazio
3 // non preceduto da + o da -
4 #include <stdio.h>
5
6 int main(void) {
7     printf(" %d\n% d\n", 547, -547);
8 }

```

```

547
-547

```

Figura 9.9 Uso del flag spazio.

9.9.4 Usare il flag

Il programma della Figura 9.10 usa il flag # per premettere 0 al valore ottale e 0x e 0X ai valori esadecimali, e inoltre per forzare la stampa del punto decimale in un valore stampato con g.

```

1 // fig09_10.c
2 // Uso del flag # con specificatori di conversione
3 // o, x, X e qualsiasi specificatore di valori in virgola mobile
4 #include <stdio.h>
5
6 int main(void) {
7     int c = 1427; // inizializza c
8     printf("%#o\n", c);
9     printf("%#x\n", c);
10    printf("%#X\n", c);
11
12    double p = 1427.0; // inizializza p
13    printf("\n%g\n", p);
14    printf("%#g\n", p);
15 }
```

```
02623
0x593
0X593
```

```
1427
1427.00
```

Figura 9.10 Uso del flag # con specificatori di conversione.

9.9.5 Usare il flag 0

Il programma della Figura 9.11 combina il flag + e il flag 0 (zero) per stampare 452 in un campo di nove spazi con un segno + e zeri iniziali, poi stampa di nuovo 452 usando soltanto il flag 0 e un campo di nove spazi.

```

1 // fig09_11.c
2 // Uso del flag 0 (zero)
3 #include <stdio.h>
4
5 int main(void) {
6     printf("%+09d\n", 452);
7     printf("%09d\n", 452);
8 }
```

```
+00000452
000000452
```

Figura 9.11 Uso del flag 0 (zero).

✓ Autovalutazione

1. (*Scelta multipla*) Quale flag di formato di `printf` è descritto con “stampa come prefisso il segno più per i valori positivi e il segno meno per i valori negativi”?

- a) -.
- b) +.
- c) 0 (zero).
- d) Nessuno dei precedenti.

Risposta: b.

2. (*Cosa fa questo codice?*) Mostra esattamente cosa stampa il seguente codice:

```
puts("1234567890123456789012345678901234567890");
printf("%10s%10d%10c%10f\n\n", "C18", 9, 'g', 6.41);
puts("1234567890123456789012345678901234567890");
printf("%-10s%-10d%-10c%-10f\n", "C18", 9, 'g', 6.41);
```

Risposta:

```
1234567890123456789012345678901234567890
C18          9           g   6.410000
```

```
1234567890123456789012345678901234567890
C18          9           g   6.410000
```

3. (*Cosa fa questo codice?*) Mostra esattamente cosa stampa il seguente codice:

```
printf("%d\n%d\n", 437, -437);
printf("%+d\n%+d\n", 437, -437);
```

Risposta:

```
437
-437
+437
-437
```

9.10 Stampa di letterali e sequenze di escape

Come avete visto nel corso del libro, i caratteri letterali inclusi nelle stringhe di controllo del formato sono semplicemente stampati da `printf`. Tuttavia, vi sono diversi caratteri che sono un “problema”, come le virgolette (") che delimitano la stessa stringa di controllo del formato. Vari caratteri di controllo, come *newline* e *tab*, devono essere rappresentati con sequenze di escape. Una sequenza di escape è rappresentata da un backslash (\), seguito da un particolare carattere di escape. La seguente tabella elenca le sequenze di escape e le azioni che queste causano.

Sequenza di escape	Descrizione
\' (virgoletta singola)	Stampa il carattere di virgoletta singola ('').
\\" (virgolette doppie)	Stampa il carattere di virgolette doppie ("").
\? (punto interrogativo)	Stampa il carattere di punto interrogativo (?).
\\\ (backslash)	Stampa il carattere di backslash (\).
\a (messaggio di avviso o squillo)	Causa l'emissione di un segnale acustico o visivo (di solito con un segnale luminoso nella finestra di esecuzione del programma).
\b (backspace)	Sposta il cursore indietro di una posizione sulla riga corrente.

\f (pagina nuova o avanzamento pagina)	Sposta il cursore all'inizio della successiva pagina logica.
\n (newline)	Sposta il cursore all'inizio della riga <i>successiva</i> .
\r (ritorno a capo)	Sposta il cursore all'inizio della riga <i>corrente</i> .
\t (tab orizzontale)	Sposta il cursore alla posizione del tab orizzontale successivo.
\v (tab verticale)	Sposta il cursore alla posizione del tab verticale successivo.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è *falsa*?
- I caratteri letterali inclusi nella stringa di controllo del formato sono ignorati da `printf`.
 - Vari caratteri di controllo, come newline e tab, devono essere rappresentati da sequenze di escape.
 - Una sequenza di escape è rappresentata da un backslash (\), seguito da un particolare carattere di escape.
 - Tutte le istruzioni precedenti sono *vere*.

Risposta: a) è *falsa*. In realtà, `printf` stampa qualsiasi carattere letterale incluso nella stringa di controllo del formato.

2. (*Scelta multipla*) Quale sequenza di escape è descritta da “Causa l’emissione di un segnale acustico o visivo (di solito con un segnale luminoso nella finestra di esecuzione del programma)”?

- \b.
- \r.
- \a.
- \v.

Risposta: \a.

9.11 Input formattato con scanf

Una precisa *formattazione dell’input* può essere realizzata con `scanf`. Ogni istruzione `scanf` contiene una stringa di controllo del formato che descrive il formato dei dati da inserire. La stringa di controllo del formato consiste di specificatori di conversione e caratteri letterali. La funzione `scanf` presenta le seguenti funzionalità di formattazione dell’input:

- Leggere tutti i tipi di dati.
- Leggere caratteri specifici da uno stream di input.
- Saltare caratteri specifici nello stream di input.

9.11.1 Sintassi di scanf

La funzione `scanf` viene scritta nella forma seguente:

```
scanf(stringa-di-controllo-del-formato, altri-argomenti);
```

La *stringa-di-controllo-del-formato* descrive i formati dell’input, e *altri-argomenti* sono puntatori a variabili nelle quali sarà memorizzato l’input.

Quando leggete dei dati, stampate un prompt di richiesta all’utente di un dato o di una porzione di dati alla volta. Evitate di chiedere all’utente di inserire molti dati in risposta a una singola richiesta. Considerate sempre cosa faranno l’utente e il vostro programma quando saranno inseriti dati scorretti, per esempio un valore per un intero che non ha senso nel contesto di un programma o una stringa senza punteggiatura o spazi.

9.11.2 Specificatori di conversione di scanf

La tabella seguente riepiloga gli specificatori di conversione usati per leggere tutti i tipi di dati. Osservate che gli specificatori di conversione **d** e **i** hanno significati differenti per l'input con **scanf**, mentre sono intercambiabili per l'output con **printf**.

Specificatore di conversione	Descrizione
Interi	
d	Legge un intero decimale con o senza segno. L'argomento corrispondente è un puntatore a un int .
i	Legge un intero decimale, ottale o esadecimale con o senza segno. L'argomento corrispondente è un puntatore a un int .
o	Legge un intero ottale. L'argomento corrispondente è un puntatore a un int senza segno.
u	Legge un intero decimale senza segno. L'argomento corrispondente è un puntatore a un int senza segno.
x o X	Legge un intero esadecimale. L'argomento corrispondente è un puntatore a un int senza segno.
h, l e ll	Vanno posti prima di uno qualsiasi degli specificatori di conversione di interi per indicare che si deve leggere, rispettivamente, un intero short , long o long long .
Numeri in virgola mobile	
e, E, f, g o G	Legge un valore in virgola mobile. L'argomento corrispondente è un puntatore a una variabile in virgola mobile.
l o L	Vanno posti prima di uno qualsiasi degli specificatori di conversione di numeri in virgola mobile per indicare che si deve leggere un valore double o long double . L'argomento corrispondente è un puntatore a una variabile double o long double .
Caratteri e stringhe	
c	Legge un carattere. L'argomento corrispondente è un puntatore a char ; non viene aggiunto alcun carattere nullo ('\\0').
s	Legge una stringa. L'argomento corrispondente è un puntatore a un array di tipo char , grande abbastanza da contenere la stringa e un carattere nullo di terminazione ('\\0') che viene aggiunto automaticamente.
Insieme di scansione	
[caratteri per la scansione]	Esegue la scansione di una stringa per un insieme di caratteri memorizzati in un array.
Altri	
p	Legge un indirizzo nello stesso formato prodotto da un'istruzione printf con lo specificatore di formato %p .
n	Memorizza il numero di caratteri letti fino a quel punto nella chiamata corrente a scanf . L'argomento corrispondente è un puntatore a un int .
%	Ignora un segno di percentuale (%) nell'input.

9.11.3 Leggere interi

Il programma della Figura 9.12 legge valori interi con vari specificatori di conversione di interi e stampa gli interi come numeri decimali. Lo specificatore di conversione %i legge interi decimali, ottali ed esadecimali.

```

1 // fig09_12.c
2 // Lettura di input con specificatori di conversione di interi
3 #include <stdio.h>
4
5 int main(void) {
6     int a = 0;
7     int b = 0;
8     int c = 0;
9     int d = 0;
10    int e = 0;
11    int f = 0;
12    int g = 0;
13
14    puts("Enter seven integers: ");
15    scanf("%d%i%ix%ox%lx", &a, &b, &c, &d, &e, &f, &g);
16
17    puts("\nThe input displayed as decimal integers is:");
18    printf("%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g);
19 }
```

```

Enter seven integers:
-70 -70 070 0x70 70 70 70

The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

Figura 9.12 Lettura di input con specificatori di conversione di interi.

9.11.4 Leggere numeri in virgola mobile

Quando si leggono numeri in virgola mobile, si può usare uno qualunque degli specificatori di conversione di numeri in virgola mobile e, E, f, g o G. Il programma della Figura 9.13 legge tre numeri in virgola mobile, uno per ciascuno dei tre tipi di specificatori di conversione, e stampa tutti e tre i numeri con lo specificatore di conversione f.

```

1 // fig09_13.c
2 // Lettura di input con specificatori di conversione di numeri in virgola mobile
3 #include <stdio.h>
4
5 int main(void) {
6     double a = 0.0;
7     double b = 0.0;
8     double c = 0.0;
9
10    puts("Enter three floating-point numbers:");
11    scanf("%le%lf%lg", &a, &b, &c);
12
13    puts("\nUser input displayed in plain floating-point notation:");
14    printf("%f\n%f\n%f\n", a, b, c);
15 }
```

```

Enter three floating-point numbers:
1.27987 1.27987e+03 3.38476e-06

User input displayed in plain floating-point notation:
1.279870
1279.870000
0.000003

```

Figura 9.13 Lettura di input con specificatori di conversione di numeri in virgola mobile.

9.11.5 Leggere caratteri e stringhe

Caratteri e stringhe vengono letti usando, rispettivamente, gli specificatori di conversione `c` e `s`. Il programma della Figura 9.14 richiede all’utente l’inserimento di una stringa. Il programma riceve in ingresso il primo carattere della stringa con `%c` e lo memorizza nella variabile di tipo carattere `x`, poi legge il resto della stringa con `%s` e lo immagazzina nell’array di caratteri `y`.

```

1 // fig09_14.c
2 // Lettura di caratteri e stringhe
3 #include <stdio.h>
4
5 int main(void) {
6     char x = '\0';
7     char y[9] = "";
8
9     printf("%s", "Enter a string: ");
10    scanf("%c%s", &x, y);
11
12    printf("The input was '%c' and \"%s\"\n", x, y);
13 }

```

```

Enter a string: Sunday
The input was 'S' and "unday"

```

Figura 9.14 Lettura di caratteri e stringhe.

9.11.6 Usare insiemi di scansione

Una sequenza di caratteri può essere letta usando un **insieme di scansione**, ovvero un insieme di caratteri racchiusi fra parentesi quadre, `[]`, e preceduto da un segno di percentuale nella stringa di controllo del formato. Un insieme di scansione esamina i caratteri nello stream di input, cercando solamente i caratteri corrispondenti ai caratteri contenuti nell’insieme di scansione. Ogni volta che viene trovato un carattere, esso è memorizzato nell’argomento array di caratteri corrispondente dell’insieme di scansione. La lettura di caratteri termina quando `scanf` incontra un carattere non contenuto nell’insieme di scansione. Se il primo carattere nello stream di input non corrisponde a un carattere nell’insieme di scansione, `scanf` non modifica il suo argomento array corrispondente. Il programma della Figura 9.15 usa l’insieme di scansione `[aeiou]` per eseguire la scansione dello stream di input alla ricerca di vocali. Per il nostro input “ooeeooahah”, vengono lette le prime sette lettere. L’ottava lettera (`h`) non è nell’insieme di scansione e pertanto `scanf` interrompe la scansione dei caratteri.

```

1 // fig09_15.c
2 // Uso di un insieme di scansione
3 #include <stdio.h>
4
5 int main(void) {

```

```

6     char z[9] = "";
7
8     printf("%s", "Enter string: ");
9     scanf("%8[aeiou]", z); // cerca un insieme di caratteri
10
11    printf("The input was \"%s\"\n", z);
12 }
```

```

Enter string: ooeeeooahah
The input was "ooeeeooa"
```

Figura 9.15 Uso di un insieme di scansione.

Invertire l'insieme di scansione

Un insieme di scansione invertito può eseguire la scansione di caratteri *non* contenuti nell'insieme di scansione. Per creare un insieme di scansione invertito, mettete un **accento circonflesso** (^) dentro le parentesi quadre prima dei caratteri per i quali eseguire la scansione. Quando si incontra un carattere contenuto nell'insieme di scansione invertito, l'input termina. Il programma della Figura 9.16 usa l'insieme di scansione invertito [^aeiou] per cercare "non-vocali".

```

1 // fig09_16.c
2 // Uso di un insieme di scansione invertito
3 #include <stdio.h>
4
5 int main(void) {
6     char z[9] = "";
7
8     printf("%s", "Enter a string: ");
9     scanf("%8[^aeiou]", z); // insieme di scansione invertito
10
11    printf("The input was \"%s\"\n", z);
12 }
```

```

Enter a string: String
The input was "Str"
```

Figura 9.16 Uso di un insieme di scansione invertito.

9.11.7 Usare larghezze di campo

È possibile usare un valore di larghezza di campo in uno specificatore di conversione per `scanf` per leggere un numero specifico di caratteri dallo stream di input. Il programma della Figura 9.17 legge una serie di cifre consecutive come un intero di due cifre e un intero formato dalle restanti cifre nello stream di input.

```

1 // fig09_17.c
2 // Lettura di dati con una larghezza di campo
3 #include <stdio.h>
4
5 int main(void) {
6     int x = 0;
7     int y = 0;
8
9     printf("%s", "Enter a six digit integer: ");
10    scanf("%2d%d", &x, &y);
```

```

11
12     printf("The integers input were %d and %d\n", x, y);
13 }

```

```

Enter a six digit integer: 123456
The integers input were 12 and 3456

```

Figura 9.17 Lettura di dati con una larghezza di campo.

9.11.8 Tralasciare caratteri in uno stream di input

Talvolta è necessario tralasciare dei caratteri nello stream di input. I caratteri di spaziatura – come spazi, new-line e tab – all'inizio di una stringa di controllo del formato ignorano tutti i caratteri di spaziatura iniziali. Altri caratteri letterali ignorano questi caratteri in specifiche posizioni dell'input. Per esempio, il vostro programma potrebbe inserire una data come

```
11-10-1999
```

Ogni numero della data deve essere memorizzato, ma i trattini che separano i numeri possono essere scartati. Per eliminare caratteri non necessari, includeteli nella stringa di controllo del formato di `scanf`. Per esempio, per scartare i trattini nell'input, usate l'istruzione

```
scanf("%d-%d-%d", &month, &day, &year);
```

Carattere di soppressione dell'assegnazione

Sebbene la precedente `scanf` elimini i trattini nell'input, è possibile che la data venga inserita come

```
10/11/1999
```

In questo caso, la `scanf` precedente non eliminerebbe i caratteri superflui. Per questa ragione, `scanf` prevede il carattere ***** di soppressione dell'assegnazione. Questo carattere permette a `scanf` di leggere qualunque tipo di dato dall'input e di scartarlo senza assegnarlo a una variabile. Il programma della Figura 9.18 usa il carattere di soppressione dell'assegnazione nella specifica di conversione `%c` per indicare che un carattere che compare nello stream di input va letto ed eliminato. Solo il mese, il giorno e l'anno vengono memorizzati. I valori delle variabili sono stampati per dimostrare che di fatto sono stati letti correttamente. Le liste di argomenti per ogni chiamata di `scanf` non contengono variabili per gli specificatori di conversione "`/*c`" contenenti il carattere di soppressione dell'assegnazione. I caratteri corrispondenti sono semplicemente eliminati.

```

1 // fig09_18.c
2 // Lettura ed eliminazione di caratteri dallo stream di input
3 #include <stdio.h>
4
5 int main(void) {
6     int month = 0;
7     int day = 0;
8     int year = 0;
9     printf("%s", "Enter a date in the form mm-dd-yyyy: ");
10    scanf("%d/*c%d/*c%d", &month, &day, &year);
11    printf("month = %d day = %d year = %d\n\n", month, day, year);
12
13    printf("%s", "Enter a date in the form mm/dd/yyyy: ");
14    scanf("%d/*c%d/*c%d", &month, &day, &year);
15    printf("month = %d day = %d year = %d\n", month, day, year);
16 }

```

```
Enter a date in the form mm-dd-yyyy: 07-04-2021
month = 7 day = 4 year = 2021

Enter a date in the form mm/dd/yyyy: 01/01/2021
month = 1 day = 1 year = 2021
```

Figura 9.18 Lettura ed eliminazione di caratteri dallo stream di input.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) Un insieme di scansione esamina i caratteri nello stream di input, cercando solamente i caratteri corrispondenti ai caratteri contenuti nell'insieme di scansione.
- b) Ogni volta che viene trovato un carattere, esso è memorizzato nell'argomento array di caratteri corrispondente dell'insieme di scansione.
- c) La lettura di caratteri termina quando `scanf` incontra un carattere non contenuto nell'insieme di scansione.
- d) Se il primo carattere dello stream corrisponde a un carattere dell'insieme di scansione, `scanf` non modifica il suo argomento array corrispondente.

Risposta: d) è *falsa*. In realtà, se il primo carattere dello stream di input *non corrisponde* a un carattere dell'insieme di scansione, `scanf` non modifica il suo argomento array corrispondente.

2. (*Completare*) È possibile usare un valore di _____ in uno specificatore di conversione per `scanf` per leggere un numero specifico di caratteri dallo stream di input.

Risposta: larghezza di campo.

3. (*Completare*) Il carattere _____ di `scanf` permette a `scanf` di leggere qualunque tipo di dato dall'input e di scartarlo senza assegnarlo a una variabile.

Risposta: * di soppressione dell'assegnazione.



9.12 Programmazione sicura in C

Il C standard elenca molti casi in cui l'uso di argomenti scorretti per funzioni di libreria può produrre comportamenti indefiniti. Questi possono provocare vulnerabilità della sicurezza, per cui vanno evitati. Tali problemi possono verificarsi quando si usa `printf` (o una qualunque delle sue varianti, come `sprintf`, `fprintf`, `printf_s`, ecc.) con specifiche di conversione scritte in modo improprio. La regola FIO00.C del CERT (<https://wiki.sei.cmu.edu/>) esamina questi problemi e presenta una tabella che mostra le combinazioni valide per i flag di formattazione, i modificatori di lunghezza e i caratteri specificatori di conversione che possono essere usati per formare specifiche di conversione. La tabella mostra anche il tipo di argomento adatto per ogni specifica di conversione valida. Quando studiate un linguaggio di programmazione, se la specifica del linguaggio dice che fare qualcosa può portare a un comportamento indefinito, evitate di farlo per prevenire vulnerabilità della sicurezza.

✓ Autovalutazione

1. (*Vero/Falso*) I comportamenti indefiniti possono provocare vulnerabilità della sicurezza, quindi è opportuno evitarli.

Risposta: Vero.

9.13 Riepilogo

Paragrafo 9.2 Stream

- Tutte le operazioni di input e output vengono effettuate tramite stream, che sono sequenze di byte.
- Lo **stream standard input** è connesso alla tastiera e gli **stream standard output** e **standard error** sono connessi allo schermo del computer.
- I sistemi operativi spesso permettono di **reindirizzare** gli stream standard ad altri dispositivi.

Paragrafo 9.3 Formattazione dell'output con printf

- Una **stringa di controllo del formato** descrive i formati in cui vengono stampati i valori di output. È costituita da **specificatori di conversione**, **flag**, **larghezze di campo**, **precisioni** e **caratteri letterali**.
- Una **specifica di conversione** è composta da un segno di percentuale % e uno specificatore di conversione.

Paragrafo 9.4 Stampa di interi

- Gli interi sono stampati con i seguenti specificatori di conversione: **d** o **i** per interi con o senza segno, **o** per interi senza segno in forma ottale, **u** per interi senza segno in forma decimale e **x** o **X** per interi senza segno in forma esadecimale. Il modificatore **h**, **l** o **ll** viene anteposto agli specificatori di conversione precedenti per indicare, rispettivamente, un intero **short**, **long** o **long long**.

Paragrafo 9.5 Stampa di numeri in virgola mobile

- I valori in virgola mobile sono stampati con i seguenti specificatori di conversione: **e** o **E** per la notazione esponenziale, **f** per la notazione regolare di numeri in virgola mobile e **g** o **G** per entrambe le notazioni e (**E**) **f**. Quando è indicato lo specificatore di conversione **g** (**G**), si usa lo specificatore di conversione **e** (**E**) se l'esponente del valore è minore di -4 o maggiore o uguale alla precisione con cui il valore è stampato.
- La **precisione** per gli specificatori di conversione **g** e **G** indica il numero massimo di cifre significative stampate.

Paragrafo 9.6 Stampa di stringhe e caratteri

- Lo specificatore di conversione **c** stampa un **carattere**.
- Lo specificatore di conversione **s** stampa una **stringa di caratteri** che termina col carattere nullo.

Paragrafo 9.7 Altri specificatori di conversione

- Lo specificatore di conversione **p** stampa un **indirizzo** in un modo definito dall'implementazione (su molti sistemi si usa la notazione esadecimale).
- Lo specificatore di conversione **%%** fa sì che venga stampato un letterale %.

Paragrafo 9.8 Stampare con larghezza di campo e precisione

- Se la **larghezza di campo** è maggiore dell'oggetto che viene stampato, per impostazione predefinita l'oggetto è **allineato a destra**.
- Le **larghezze di campo** possono essere usate con tutti gli specificatori di conversione.
- La **precisione** usata con gli specificatori di conversione di interi indica il numero minimo di cifre stampate.
- La **precisione** usata con gli specificatori di conversione di numeri in virgola mobile **e**, **E** e **f** indica il numero di cifre che compaiono dopo il punto decimale. La precisione usata con gli specificatori di conversione di numeri in virgola mobile **g** e **G** indica il numero di cifre significative da stampare.

- La **precisione** usata con lo specificatore di conversione `s` indica il numero di caratteri da stampare.
- La **larghezza di campo** e la **precisione** possono essere combinate mettendo tra il segno di percentuale e lo specificatore di conversione prima la larghezza di campo, poi un punto decimale e infine la precisione.
- È possibile specificare la **larghezza di campo** e la **precisione** con **espressioni intere** nella lista di argomenti che segue la stringa di controllo del formato. Per fare così, usate un asterisco (*) per la larghezza di campo o la precisione. L'argomento corrispondente nella lista di argomenti viene usato al posto dell'asterisco.

Paragrafo 9.9 Flag di formato di printf

- Il **flag** - allinea il suo argomento a sinistra in un campo.
- Il **flag +** stampa un segno più per i valori positivi e un segno meno per i valori negativi.
- Il **flag spazio** stampa uno spazio che precede un valore positivo che non viene stampato con il flag +.
- Il **flag #** antepone 0 ai valori ottali e 0x o 0X ai valori esadecimali e forza la stampa del punto decimale per i valori in virgola mobile stampati con e, E, f, g o G.
- Il **flag 0** stampa zeri iniziali per un valore che non occupa la sua intera larghezza di campo.

Paragrafo 9.10 Stampa di letterali e sequenze di escape

- La maggior parte dei **caratteri letterali** da stampare in un'istruzione `printf` può essere semplicemente inclusa nella stringa di controllo del formato. Tuttavia, vi sono diversi caratteri "problematici", come il carattere di doppie virgolette ("") che delimita la stessa stringa di controllo del formato. Vari **caratteri di controllo**, come newline e tab, devono essere rappresentati con **sequenze di escape**. Una sequenza di escape è rappresentata con un backslash (\) seguito da un particolare carattere di escape.

Paragrafo 9.11 Input formattato con scanf

- Una precisa **formattazione dell'input** può essere realizzata con la funzione di libreria `scanf`.
- Gli interi vengono letti con `scanf` usando gli specificatori di conversione `d` e `i` per gli interi con o senza segno e `o`, `u`, `x` o `X` per gli interi senza segno nei formati ottale, decimale ed esadecimale. I modificatori `h`, `l` o `ll` sono posti prima di uno specificatore di conversione di interi per leggere, rispettivamente, un intero `short`, `long` o `long long`.
- I valori in virgola mobile sono letti con `scanf` usando gli specificatori di conversione `e`, `E`, `f`, `g` o `G`. I modificatori `l` e `L` sono posti prima di uno qualsiasi degli specificatori di conversione di numeri in virgola mobile per indicare che il valore in input è un valore, rispettivamente, `double` o `long double`.
- I caratteri sono letti con `scanf` con lo specificatore di conversione `c`.
- Le stringhe sono lette con `scanf` con lo specificatore di conversione `s`.
- Una `scanf` con un **insieme di scansione** esegue la scansione dei caratteri in input, cercando solo i caratteri corrispondenti ai caratteri contenuti nell'insieme di scansione. Quando viene trovato uno di questi caratteri, esso viene memorizzato in un array di caratteri. Quando si incontra un carattere non contenuto nell'insieme di scansione, la lettura di caratteri termina.
- Per creare un **insieme di scansione invertito**, mettete un accento circonflesso (^) dentro le parentesi quadre prima dei caratteri per cui effettuare la scansione. `scanf` memorizza i caratteri che non compiono nell'insieme di scansione invertito e si ferma quando incontra un carattere contenuto nell'insieme di scansione invertito.
- `scanf` legge i **valori degli indirizzi** con lo specificatore di conversione `p`.
- Lo specificatore di conversione `n` memorizza il **numero di caratteri letti** fino a quel momento nella `scanf` corrente. L'argomento corrispondente è un puntatore a `int`.
- Il **carattere di soppressione dell'assegnazione** (*) legge i dati dallo stream di input e li elimina.
- Una **larghezza di campo** si usa in `scanf` per leggere un numero specifico di caratteri dallo stream di input.

Esercizi di autovalutazione

- 9.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
- Tutto l'input e l'output è trattato in forma di _____.
 - Lo stream _____ è normalmente connesso alla tastiera.
 - Lo stream _____ è normalmente connesso allo schermo del computer.
 - Una precisa formattazione dell'output si ottiene con la funzione _____.
 - La stringa di controllo del formato può contenere _____, _____, _____, _____ e _____.
 - Lo specificatore di conversione _____ o _____ può essere usato per stampare un intero decimale con segno.
 - Gli specificatori di conversione _____, _____ e _____ sono usati per stampare interi senza segno in forma, rispettivamente, ottale, decimale ed esadecimale.
 - I modificatori _____ e _____ sono messi prima degli specificatori di conversione di interi per stampare valori interi short o long.
 - Lo specificatore di conversione _____ è usato per stampare un valore in virgola mobile con notazione esponenziale.
 - Il modificatore _____ viene messo prima di uno specificatore di conversione di un numero in virgola mobile per indicare che deve essere stampato un valore long double.
 - Gli specificatori di conversione e, E e f sono stampati con _____ cifre di precisione alla destra del punto decimale se non è specificata alcuna precisione.
 - Gli specificatori di conversione _____ e _____ sono usati per stampare, rispettivamente, stringhe e caratteri.
 - Tutte le stringhe terminano con il carattere _____.
 - La larghezza di campo e la precisione in uno specificatore di conversione per printf possono essere controllate con espressioni intere ponendo un _____ per la larghezza di campo o per la precisione e mettendo un'espressione intera nell'argomento corrispondente della lista di argomenti.
 - Il flag _____ fa sì che l'output sia allineato a sinistra in un campo.
 - Il flag _____ fa sì che i valori siano stampati o con un segno più o con un segno meno.
 - Una precisa formattazione dell'input viene realizzata con la funzione _____.
 - Un _____ serve per fare la scansione di una stringa per caratteri specifici da memorizzare in un array.
 - Lo specificatore di conversione _____ può essere usato per leggere interi ottali, decimali ed esadecimali con o senza segno.
 - Gli specificatori di conversione _____ possono essere usati per leggere un valore double.
 - Il _____ è usato per leggere dati dallo stream di input ed eliminarli senza assegnarli a una variabile.
 - Una _____ può essere usata in uno specificatore di conversione per scanf per indicare che si deve leggere un numero specifico di caratteri o di cifre dallo stream di input.
- 9.2 Trovate l'errore in ognuna delle seguenti istruzioni e spiegate come sarebbe possibile correggerlo.
- L'istruzione seguente deve stampare il carattere 'c'.

```
printf("%s\n", 'c');
```
 - L'istruzione seguente deve stampare 9.375%.

```
printf("%.3f%", 9.375);
```
 - L'istruzione seguente deve stampare il primo carattere di "Monday".

```
printf("%c\n", "Monday");
```
 - `puts("A string in quotes");`
 - `printf(%d%d, 12, 20);`
 - `printf("%c", "x");`
 - `printf("%s\n", 'Richard');`

- 9.3 Scrivete un'istruzione per ognuna delle seguenti operazioni.
- Stampare 1234 allineato a destra in un campo di 10 cifre.
 - Stampare 123.456789 in notazione esponenziale con un segno (+ o -) e 3 cifre di precisione.
 - Leggere un valore `double` e memorizzarlo nella variabile `number`.
 - Stampare 100 in forma ottale preceduto da 0.
 - Leggere una stringa e memorizzarla nell'array di caratteri `string`.
 - Leggere caratteri e memorizzarli nell'array `n` finché non si incontra un carattere diverso da una cifra.
 - Usare le variabili intere `x` e `y` per specificare la larghezza di campo e la precisione usate per stampare il valore `double` 87.4573.
 - Leggere un valore della forma 3.5%. Memorizzare il valore della percentuale nella variabile `float percent` ed eliminare il % dallo stream di input. Non usare il carattere di soppressione dell'assegnazione.
 - Stampare 3.333333 come valore `long double` con un segno (+ o -) in un campo di 20 caratteri con una precisione di 3.

Risposte agli esercizi di autovalutazione

- 9.1 a) stream. b) standard input. c) standard output. d) `printf`. e) specificatori di conversione, flag, larghezze di campo, precisioni, caratteri letterali. f) d, i, g) o, u, x (o X). h) h, l, i) e (o E). j) L. k) 6. l) s, c, m) NULL ('\0'). n) asterisco (*). o) - (meno). p) + (più). q) `scanf`. r) insieme di scansione. s) i, t) le, lE, lf, lg o lG. u) carattere di soppressione dell'assegnazione (*). v) larghezza di campo.
- 9.2 a) Errore: lo specificatore di conversione `s` si aspetta un argomento di tipo puntatore a `char`.
Correzione: per stampare il carattere 'c', usate lo specificatore di conversione `%c` oppure cambiate 'c' in "c".
- b) Errore: cercare di stampare il carattere letterale % senza usare lo specificatore di conversione `%%`.
Correzione: usate `%%` per stampare un carattere letterale %.
- c) Errore: lo specificatore di conversione `c` si aspetta un argomento di tipo `char`.
Correzione: per stampare il primo carattere di "Monday" usate lo specificatore di conversione `%1s`.
- d) Errore: cercare di stampare il carattere letterale " senza usare la sequenza di escape \".
Correzione: sostituite ogni carattere di doppie virgolette all'interno della stringa con \".
- e) Errore: la stringa di controllo del formato non è racchiusa tra doppie virgolette.
Correzione: racchiudete `%d%d` tra doppie virgolette.
- f) Errore: il carattere x è racchiuso tra doppie virgolette.
Correzione: le costanti carattere da stampare con `%c` devono essere racchiuse tra virgolette singole.
- g) Errore: la stringa da stampare è racchiusa tra virgolette singole.
Correzione: usate le doppie virgolette invece delle virgolette singole per rappresentare una stringa.

- 9.3 a) `printf("%10d\n", 1234);`
b) `printf("%+.3e\n", 123.456789);`
c) `scanf("%lf", &number);`
d) `printf("%#o\n", 100);`
e) `scanf("%s", string);`
f) `scanf("%[0123456789]", n);`
g) `printf("%.*f\n", x, y, 87.4573);`
h) `scanf("%f%%", &percent);`
i) `printf("%+20.3Lf\n", 3.333333);`

Esercizi

- 9.4 Scrivete un'istruzione `printf` o `scanf` per ognuna delle seguenti operazioni.
- Stampare l'intero senza segno 40000 allineato a sinistra in un campo di 15 cifre con 8 cifre.
 - Leggere un valore esadecimale e memorizzarlo nella variabile `hex`.
 - Stampare 200 con e senza segno.

- d) Stampare 100 nella forma esadecimale preceduta da 0x.
 e) Leggere caratteri e memorizzarli nell'array s finché non si incontra la lettera p.
 f) Stampare 1.234 preceduto da zeri in un campo di 9 cifre.
 g) Leggere un valore temporale della forma hh:mm:ss, memorizzando le singole parti nelle variabili intere hour, minute e second. Tralasciare i due punti (:) nello stream di input. Usare il carattere di soppressione dell'assegnazione.
 h) Leggere una stringa della forma "characters" dallo standard input. Memorizzare la stringa nell'array di caratteri s. Eliminare le doppie virgolette dallo stream di input.
 i) Leggere un valore temporale della forma hh:mm:ss, memorizzando le singole parti nelle variabili intere hour, minute e second. Tralasciare i due punti (:) nello stream di input. Non usare il carattere di soppressione dell'assegnazione.
- 9.5 Mostrate cosa stampa ognuna delle seguenti istruzioni. Se un'istruzione è scorretta, indicate perché.
- `printf("%-10d\n", 10000);`
 - `printf("%c\n", "This is a string");`
 - `printf("%.*lf\n", 8, 3, 1024.987654);`
 - `printf("%#o\n%#X\n%#e\n", 17, 17, 1008.83689);`
 - `printf("% 1d\n%+1d\n", 1000000, 1000000);`
 - `printf("%10.2E\n", 444.93738);`
 - `printf("%10.2g\n", 444.93738);`
 - `printf("%d\n", 10.987);`
- 9.6 Trovate l'errore (o gli errori) in ognuno dei seguenti segmenti di programma. Spiegate come ogni errore può essere corretto.
- `printf("%s\n", 'Happy Birthday');`
 - `printf("%c\n", 'Hello');`
 - `printf("%c\n", "This is a string");`
 - L'istruzione seguente deve stampare "Bon Voyage":
`printf("%s", "Bon Voyage");`
 - `char day[] = "Sunday";`
`printf("%s\n", day[3]);`
 - `puts('Enter your name: '');`
 - `printf(%f, 123.456);`
 - L'istruzione seguente deve stampare i caratteri '0' e 'K':
`printf("%s%s\n", '0', 'K');`
 - `char s[10];`
`scanf("%c", s[7]);`

9.7 (*Differenze tra %d e %i*) Scrivete un programma per verificare la differenza tra gli speciatori di conversione %d e %i quando sono usati in istruzioni scanf. Chiedete all'utente di inserire due interi separati da uno spazio. Usate le istruzioni

```
scanf("%i%d", &x, &y);
printf("%d %d\n", x, y);
```

per leggere e stampare i valori. Testate il programma con i seguenti insiemi di dati di input:

```
10      10
-10     -10
010     010
0x10    0x10
```

9.8 (*Stampa di numeri con varie larghezze di campo*) Scrivete un programma per verificare i risultati della stampa del valore intero 12345 e del valore in virgola mobile 1.2345 in campi di varie dimensioni. Cosa succede quando i valori sono stampati in campi contenenti meno cifre dei valori?

9.9 (Arrotondamento di numeri in virgola mobile) Scrivete un programma che stampi il valore 100.453627 arrotondato alla cifra più vicina, quale quella dei decimi, dei centesimi, dei millesimi e dei decimillesimi.

9.10 (Conversione di temperature) Scrivete un programma che converta temperature intere in gradi Fahrenheit da 0 a 212 gradi in temperature Celsius in virgola mobile con 3 cifre di precisione. Eseguite il calcolo usando la formula

```
celsius = 5.0 / 9.0 * (fahrenheit - 32);
```

L'output va stampato in due colonne allineate a destra di 10 caratteri ciascuna e le temperature Celsius vanno precedute da un segno sia per valori positivi che per quelli negativi.

9.11 (Sequenze di escape) Scrivete un programma per provare le sequenze di escape \', \", \?, \\, \a, \b, \n, \r e \t. Per le sequenze di escape che spostano il cursore, stampate un carattere prima e dopo la stampa della sequenza di escape, così è chiaro dove è stato spostato il cursore.

9.12 (Stampa di un punto interrogativo) Scrivete un programma che determini se ? può essere stampato come un carattere letterale facente parte di una stringa di controllo del formato di printf invece di usare la sequenza di escape \?.

9.13 (Leggere un intero con ogni specificatore di conversione per scanf) Scrivete un programma che legga il valore 437 usando ognuno degli specificatori di conversione per scanf. Stampate ogni valore letto usando tutti gli specificatori di conversione di interi.

9.14 (Stampare un numero con gli specificatori di conversione di numeri in virgola mobile) Scrivete un programma che usi ognuno degli specificatori di conversione e, f e g per leggere il valore 1.2345. Stampate il valore di ogni variabile per provare che è possibile usare ognuno degli specificatori di conversione per leggere questo stesso valore.

9.15 (Leggere stringhe tra virgolette) In alcuni linguaggi di programmazione le stringhe sono inserite circondate da virgolette singole o da virgolette doppie. Scrivete un programma che legga le tre stringhe suzy, "suzy" e 'suzy'. Le virgolette singole e doppie sono ignorate dal C o lette come parte della stringa?

9.16 (Stampare un punto interrogativo come una costante carattere) Scrivete un programma che determini se è possibile stampare ? come la costante carattere '?' invece che con la sequenza di escape della costante carattere '\?'. Usate lo specificatore di conversione %c nella stringa di controllo del formato di un'istruzione printf.

9.17 (Utilizzare %g con varie precisioni) Scrivete un programma che usi lo specificatore di conversione g per stampare il valore 9876.12345. Stampate il valore con precisioni nell'intervallo da 1 a 9.

CAPITOLO

10

Sommario del capitolo

- 10.1 Introduzione
- 10.2 Definizione di strutture
- 10.3 Inizializzazione di strutture
- 10.4 Accesso ai membri delle strutture con . e ->
- 10.5 Uso delle strutture con le funzioni
- 10.6 `typedef`
- 10.7 Caso pratico di simulazione di numeri casuali: mescolamento e distribuzione di carte ad alte prestazioni
- 10.8 Unioni
- 10.9 Operatori bit a bit
- 10.10 Campi di bit
- 10.11 Costanti di enumerazione
- 10.12 Strutture e unioni anonime
- 10.13 Programmazione sicura in C
- 10.14 Riepilogo

Strutture, unioni, manipolazione di bit ed enumerazioni

Obiettivi

- Creare e usare strutture (`struct`), unioni (`union`) ed enumerazioni (`enum`).
- Comprendere le strutture autoreferenziali.
- Apprendere le operazioni che possono essere effettuate su esempi di strutture.
- Inizializzare membri di strutture.
- Accedere a membri di strutture.
- Passare esempi di strutture a funzioni per valore e per riferimento.
- Usare la parola chiave `typedef` per creare alias per nomi di tipi esistenti.
- Apprendere le operazioni che possono essere effettuate sulle unioni.
- Inizializzare le unioni.
- Manipolare dati interi con gli operatori bit a bit (cioè applicati ai singoli bit).
- Creare campi di bit per memorizzare dati in modo compatto.
- Usare costanti di enumerazione.
- Considerare problemi di sicurezza quando si lavora con strutture, manipolazione di bit ed enumerazioni.

10.1 Introduzione

Le **strutture**, talvolta chiamate **aggregati** nel C standard, sono collezioni di variabili collegate sotto un unico nome. Le strutture possono contenere variabili di differenti tipi di dati, contrariamente agli array, che contengono solo elementi dello stesso tipo di dati. Qui analizzeremo:

- `typedef`, per creare *alias* per tipi di dati definiti in precedenza;
- le unioni (`union`), simili alle strutture, ma con membri che *condividono* lo stesso spazio di memoria;
- gli operatori bit a bit, per manipolare i bit di operandi interi;
- i campi di bit: membri `unsigned int` o `int` di strutture o di unioni per le quali si specifica il numero di bit in cui essi sono memorizzati, così da impacchettare le informazioni facilmente in modo compatto;
- le enumerazioni: insiemi di costanti intere rappresentate da identificatori.

Nei Capitoli 11 e 12, vedrete quanto segue:

- le strutture comunemente definiscono i record da archiviare in file;
- i puntatori e le strutture facilitano la formazione di strutture di dati come liste collegate, code, pile e alberi.

✓ Autovalutazione

1. (*Completare*) Le _____ sono simili alle strutture, ma con membri che condividono lo stesso spazio di memoria.

Risposta: unioni (union).

2. (*Completare*) Le _____ sono insiemi di costanti intere rappresentate da identificatori.

Risposta: enumerazioni.

10.2 Definizione di strutture

Le strutture sono **tipi di dati derivati**, essendo costruite usando oggetti di altri tipi. La parola chiave **struct** introduce la definizione di una struttura, come in

```
struct card {
    const char *face;
    const char *suit;
};
```

L'identificatore **card** è l'**etichetta della struttura**, che viene usata con **struct** per dichiarare le variabili del **tipo di struttura**, per esempio **struct card**. Le variabili dichiarate entro le parentesi di **struct** sono i **membri** della struttura. I membri di **struct** devono avere nomi unici, anche se tipi di strutture separate possono contenere membri dello stesso nome senza che ciò crei un conflitto. Ogni definizione di struttura termina con un punto e virgola.

La definizione di **struct card** contiene i membri **face** e **suit** di tipo **const char ***. I membri di una struttura possono essere variabili **const** oppure non-**const** dei tipi di dati primitivi (es. **int**, **double** ecc.) o aggregati, come array o altri oggetti di tipo **struct**. Nel Capitolo 6 abbiamo visto che gli elementi di un array sono tutti dello stesso tipo. I membri di una struttura, invece, possono essere di tipi differenti. Per esempio, la seguente definizione di **struct** contiene membri di tipo array di caratteri per il nome e il cognome di un impiegato (**employee**), un membro di tipo **int** per l'età dell'impiegato e un membro di tipo **double** per la sua paga oraria:

```
struct employee {
    char firstName[20];
    char lastName[20];
    int age;
    double hourlySalary;
};
```

10.2.1 Strutture autoreferenziali

 Un tipo di **struct** non può contenere una variabile di quello stesso tipo di **struct** (sarebbe un errore di compilazione), ma può contenere un puntatore a quel tipo di **struct**. Per esempio, la seguente definizione aggiornata di **struct employee** contiene un puntatore al manager dell'impiegato, che è un altro oggetto di tipo **struct employee**:

```
struct employee {
    char firstName[20];
    char lastName[20];
    unsigned int age;
```

```

    double hourlySalary;
    struct employee *managerPtr; // puntatore
};

```

Una struttura contenente un membro che è un puntatore allo *stesso* tipo di struttura è detta **struttura autoreferenziale**. Le strutture autoreferenziali sono usate per costruire strutture di dati collegate.

10.2.2 Definizione di variabili di tipi di struttura

Le definizioni di strutture non riservano alcuno spazio in memoria; invece, ogni definizione crea un nuovo tipo di dati usato per definire le variabili – come un progetto di come costruire esempi di quella struct. Le istruzioni seguenti riservano memoria per le variabili usando il tipo struct card:

```

struct card myCard;
struct card deck[52];
struct card *cardPtr;

```

La variabile myCard è un oggetto di tipo struct card, l'array deck è composto da 52 oggetti di tipo struct card, e cardPtr è un puntatore a un oggetto di tipo struct card.

Le variabili di un dato tipo di struttura possono anche essere definite mettendo una lista dei nomi delle variabili, separati da virgole, tra la parentesi che chiude la definizione della struttura e il punto e virgola che la termina. Per esempio, si possono accoppare le definizioni precedenti nella definizione di struct card:

```

struct card {
    const char *face;
    const char *suit;
} myCard, deck[52], *cardPtr;

```

10.2.3 Etichette delle strutture

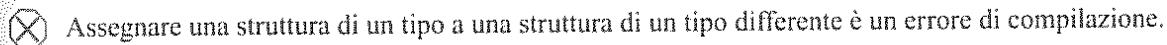
Le etichette delle strutture sono facoltative. Se una definizione di struttura non specifica un'etichetta, deve essere definita ogni variabile di quel tipo, come mostrato nel frammento di codice precedente. Fornite sempre l'etichetta di una struttura, in modo da poter dichiarare in seguito nuove variabili di quel tipo.

10.2.4 Operazioni che si possono eseguire sulle strutture

Sulle strutture si possono effettuare le operazioni seguenti:

- assegnare una variabile struct a un'altra variabile dello *stesso* tipo (Paragrafo 10.7) – per un membro puntatore, viene copiato solo l'indirizzo memorizzato nel puntatore;
- accedere all'indirizzo (&) di una variabile di tipo struct (Paragrafo 10.4);
- accedere ai membri di una variabile di tipo struct (Paragrafo 10.4);
- usare l'operatore sizeof per determinare la dimensione di una variabile struct;
- inizializzare a zero una variabile struct nella sua definizione, come in

```
struct card myCard = {};
```



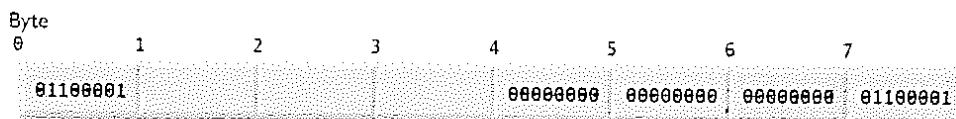
Confrontare oggetti struttura non è consentito

Le strutture *non* possono essere confrontate usando gli operatori == e !=, perché i membri della struttura potrebbero non essere memorizzati in byte di memoria consecutivi. A volte vi sono “buchi” in una struttura, perché i computer possono memorizzare tipi specifici di dati solo all'interno dei confini di certe unità di memoria come metà parola, una parola intera o una parola doppia. Questo dipende dalla macchina. Una parola è un'unità di memoria (di solito di 4 o 8 byte) usata per memorizzare dati in un computer.

Considerate la seguente definizione di struttura, che definisce anche le variabili `sample1` e `sample2`:

```
struct example {
    char c;
    int i;
} sample1, sample2;
```

Un computer con parole di quattro byte può richiedere che ogni membro di `struct example` sia allineato su un confine di parola, ossia all'inizio di una parola. Il diagramma seguente mostra un esempio di allineamento in memoria per una variabile di tipo `struct example` a cui è stato assegnato il carattere 'a' e l'intero 97. Qui mostriamo le rappresentazioni in termini di bit dei valori.



 Se ciascun membro è memorizzato allineato all'inizio di una parola, ogni variabile di tipo `struct example` ha un buco di tre byte nei byte 1-3. Il valore nel buco di tre byte è *indefinito*. Anche se i valori dei membri di `sample1` e `sample2` sono effettivamente uguali, le strutture non sono necessariamente uguali, perché è probabile che i buchi non contengano valori identici. Le dimensioni dei tipi di dati e le caratteristiche relative all'allineamento in memoria sono dipendenti dalla macchina.

✓ Autovalutazione

1. (*Scelta multipla*) Considerate la seguente definizione di `struct name`:

```
struct name {
    const char *first;
    const char *last;
};
```

Quale delle seguenti affermazioni a), b) o c) è *falsa*?

- La parola chiave `struct` introduce la definizione di una struttura.
- L'etichetta di struttura `name` può essere usata con `struct` per dichiarare variabili del tipo struttura.
- Le variabili dichiarate tra le parentesi di `struct` sono i membri della struttura, che devono avere nomi unici.
- Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

2. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) non è un'operazione valida che può essere eseguita su una struttura?

- Assegnare variabili `struct` a variabili `struct` dello stesso tipo.
- Dereferenziare una variabile `struct`.
- Accedere ai membri di una variabile `struct` e usare l'operatore `sizeof` per determinare la dimensione di una variabile `struct`.
- Tutte le affermazioni precedenti sono *vere*.

Risposta: b) non è un'operazione valida. Non è possibile dereferenziare una struttura perché non è un puntatore, ma si può accedere all'indirizzo di una struttura con l'operatore &.

10.3 Inizializzazione di strutture

Come gli array, è possibile inizializzare variabili di tipo `struct` tramite una lista di inizializzatori. Per esempio, l'istruzione seguente crea la variabile `myCard` usando il tipo `struct card` (Paragrafo 10.2) e inizializza il membro `face` a "Three" e il membro `suit` a "Hearts":

```
struct card myCard = {"Three", "Hearts"};
```

Se vi sono meno inizializzatori che membri, i restanti membri sono automaticamente inizializzati a 0 o NULL (per membri puntatore).

Le variabili di tipo struttura definite al di fuori di una definizione di funzione (cioè esternamente) sono inizializzate a 0 o NULL se non sono esplicitamente inizializzate nella definizione esterna. Si possono anche assegnare variabili di struttura ad altre variabili di struttura dello stesso tipo o assegnare valori a singoli membri della struttura.

✓ Autovalutazione

1. (*Vero/Falso*) Se nella lista vi sono meno inizializzatori dei membri nella struttura, i restanti membri non sono inizializzati.

Risposta: *Falso*. In realtà, essi sono inizializzati a 0 (o NULL se il membro è un puntatore).

2. (*Vero/Falso*) Si possono assegnare variabili di struttura ad altre variabili di struttura dello stesso tipo o assegnare valori a singoli membri della struttura.

Risposta: *Vero*.

10.4 Accesso ai membri delle strutture con . e ->

Potete accedere ai membri di una struttura con:

- l'**operatore di membro di struttura** **(.)**, chiamato anche operatore punto;
- l'**operatore di puntatore a struttura** **(->)**, o **operatore freccia**.

Operatore di membro di struttura (.)

L'operatore di membro di struttura accede a un membro di una struttura tramite il nome di una variabile di struttura.

Per esempio, usando la variabile di struttura `myCard` del Paragrafo 10.3, possiamo stampare il membro `suit` con l'istruzione:

```
printf("%s", myCard.suit); // stampa Hearts
```

Operatore di puntatore a struttura (->)

Si può accedere a un membro di una struttura tramite un puntatore alla struttura usando l'operatore di puntatore a struttura: un segno meno (-) e un segno di maggiore di (>) senza spazi intermedi. Se il puntatore `cardPtr` punta all'oggetto `myCard` di tipo `struct card` che abbiamo definito in precedenza, possiamo stampare il suo membro `suit` con l'istruzione:

```
printf("%s", cardPtr->suit); // stampa Hearts
```

L'espressione `cardPtr->suit` è equivalente a `(*cardPtr).suit`, che dereferenzia il puntatore e accede al membro `suit` usando l'operatore di membro di struttura **(.)**. Le parentesi sono qui necessarie perché l'operatore di membro di struttura **(.)** ha una precedenza più alta dell'operatore che dereferenzia il puntatore **(*)**. L'operatore di puntatore a struttura e l'operatore di membro di struttura hanno la precedenza più alta e sono associativi da sinistra a destra, insieme con le parentesi usate per chiamare le funzioni e le parentesi quadre `([])` usate per l'indicizzazione di array.

Convenzioni sugli spazi

Non mettete spazi attorno agli operatori `->` e `.` (punto) per evidenziare che le espressioni in cui sono contenuti gli operatori sono essenzialmente singoli nomi di variabili. Inserire uno spazio tra i componenti `- e >` dell'operatore di puntatore a struttura o tra i componenti di un qualsiasi altro operatore formato da una sequenza multipla di tasti (eccetto `?:`) è un errore di sintassi.



Uso degli operatori di membro di struttura e di puntatore a struttura

Il programma della Figura 10.1 fa riferimento ai membri della struttura `myCard` usando gli operatori di membro di struttura e di puntatore a struttura. Le righe 16-17 assegnano "Ace" e "Spades" ai membri di `myCard`. La riga 19 assegna l'indirizzo di `myCard` al puntatore `cardPtr`. Le righe 21-23 stampano i membri di `myCard` usando:

- l'operatore di membro di struttura e il nome della variabile `myCard`;
- l'operatore di puntatore a struttura e il puntatore `cardPtr`;
- l'operatore di membro di struttura e il puntatore `cardPtr` dereferenziato.

```

1 // fig10_01.c
2 // Operatore di membro di struttura e
3 // operatore di puntatore a struttura
4 #include <stdio.h>
5
6 // definizione della struttura card
7 struct card {
8     const char *face; // definisci il puntatore face
9     const char *suit; // definisci il puntatore suit
10 };
11
12 int main(void) {
13     struct card myCard; // definisci una variabile di tipo struct card
14
15     // inserisci le stringhe in myCard
16     myCard.face = "Ace";
17     myCard.suit = "Spades";
18
19     struct card *cardPtr = &myCard; // assegna l'indirizzo di myCard a cardPtr
20
21     printf("%s of %s\n", myCard.face, myCard.suit);
22     printf("%s of %s\n", cardPtr->face, cardPtr->suit);
23     printf("%s of %s\n", (*cardPtr).face, (*cardPtr).suit);
24 }
```

Ace of Spades

Ace of Spades

Ace of Spades

Figura 10.1 Operatore di membro di struttura e operatore di puntatore a struttura.

✓ Autovalutazione

1. (*Completare*) L'operatore di membro di struttura _____ e l'operatore di puntatore a struttura _____ possono essere usati per accedere ai membri delle strutture.

Risposta: . (punto), ->.

2. (*Vero/Falso*) L'espressione `cardPtr->suit` equivale a `(*cardPtr).suit`, che dereferenzia il puntatore e accede al membro `suit` usando l'operatore di membro di struttura. Le parentesi sono facoltative.

Risposta: *Falso*. Le espressioni, come mostrato, sono equivalenti. Le parentesi *sono* necessarie perché l'operatore di membro di struttura (.) ha una precedenza più alta dell'operatore che dereferenzia il puntatore (*).

10.5 Uso delle strutture con le funzioni

Con le strutture, è possibile passare alle funzioni:

- membri individuali di una struttura;
- interi oggetti struttura;
- puntatori a oggetti struttura.

I singoli membri di una struttura e gli interi oggetti struttura sono passati per valore, pertanto le funzioni non possono modificarli nella funzione chiamante. Per passare una struttura per riferimento, usate l'indirizzo dell'oggetto struttura.

 Passare le strutture per riferimento è più efficiente che passare le strutture per valore, cosa che richiede la copiatura dell'intera struttura. Gli array di oggetti struttura, come tutti gli altri array, sono passati automaticamente per riferimento.

Passare un array per valore

Nel Capitolo 6 abbiamo affermato che un array può essere passato per valore usando una struttura. Per farlo, create una struttura con un array come membro. Le strutture sono passate per valore, per cui i suoi membri sono passati per valore.

✓ Autovalutazione

1. (*Completare*) Gli oggetti struttura e i membri individuali di una struttura sono passati alle funzioni per _____.

Risposta: valore.

2. (*Discussione*) Come si passa un array per valore?

Risposta: Si mette semplicemente l'array in una struttura e si passa la struttura. Le strutture di norma sono passate per valore.

10.6 `typedef`

La parola chiave `typedef` consente di creare sinonimi (o alias) per tipi di dati precedentemente definiti. Viene comunemente usata per creare nomi più corti per i tipi di strutture e per semplificare le dichiarazioni di tipi quali i puntatori a funzioni. Nell'esempio seguente `typedef` definisce `Card` come un sinonimo per il tipo `struct card`:

```
typedef struct card Card;
```

Per convenzione, scrivete in maiuscolo la prima lettera dei nomi creati con `typedef` per mettere in evidenza che sono sinonimi di altri nomi di tipo.

Potete ora utilizzare `Card` per dichiarare variabili di tipo `struct card`. La dichiarazione

```
Card deck[52];
```

dichiara un array di 52 strutture `Card` (cioè variabili di tipo `struct card`).

Creare un nome nuovo con `typedef` *non* crea un nuovo tipo; `typedef` crea un nuovo *nome di tipo*, che può essere usato come *alias* per un nome di tipo esistente. Un nome significativo contribuisce a rendere il programma autodocumentante. Per esempio, quando leggiamo la dichiarazione precedente, sappiamo che "deck" è un array di 52 `Card`.

Combinazione di `typedef` con le definizioni di `struct`

I programmati spesso usano `typedef` per definire un tipo di struttura, per cui non è necessaria l'etichetta della struttura. Per esempio, la definizione seguente crea il tipo di struttura `Card`:

```
typedef struct {
    const char *face;
    const char *suit;
} Card;
```

Sinonimi per i tipi predefiniti

L'uso di `typedef` può contribuire a rendere un programma più leggibile e più facile da mantenere. Spesso si usa `typedef` per creare sinonimi per i tipi predefiniti. Per esempio, un programma che richiede interi di quattro byte può usare il tipo `int` su un sistema e il tipo `long` su un altro. I programmi progettati per la portabilità usano spesso `typedef` per creare un alias per interi di quattro byte, come `Integer`. Per rendere il programma portabile su un'altra piattaforma, potete semplicemente modificare il `typedef` di `Integer` e ricompilare il programma.

✓ Autovalutazione

1. (*Codice*) Scrivete un'istruzione `typedef` che crei il nome di tipo più breve `Dice` per il tipo di struttura `struct dice`.

Risposta: `typedef struct dice Dice;`

2. (*Vero/Falso*) Creare un nome nuovo con `typedef` crea un nuovo tipo.

Risposta: *Falso*. Creare un nome nuovo con `typedef` non crea un nuovo tipo. Crea semplicemente un nuovo *nome di tipo*, che può essere usato come *alias* per un nome di tipo esistente.

10.7 Caso pratico di simulazione di numeri casuali: mescolamento e distribuzione di carte ad alte prestazioni

Il programma della Figura 10.2 si basa sulla simulazione del mescolamento e della distribuzione di carte esaminata nel Capitolo 7. Il programma rappresenta il mazzo di carte come un array di strutture (`struct`) `Card` e usa algoritmi ad alte prestazioni per mescolare e distribuire.

```
1 // fig10_02.c
2 // Programma per mescolare e distribuire le carte con uso di strutture
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 #define CARDS 52
8 #define FACES 13
9
10 // definizione della struttura card
11 struct card {
12     const char *face; // definisci il puntatore face
13     const char *suit; // definisci il puntatore suit
14 };
15
16 typedef struct card Card; // nuovo nome di tipo per struct card
17
18 // prototipi
```

```
19 void fillDeck(Card * const deck, const char *faces[], const char *suits[]);
20 void shuffle(Card * const deck);
21 void deal(const Card * const deck);
22
23 int main(void) {
24     Card deck[CARDS]; // definisci l'array di carte
25
26     // inizializza l'array di puntatori dei valori
27     const char *faces[] = { "Ace", "Deuce", "Three", "Four", "Five",
28         "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
29
30     // inizializza l'array di puntatori dei semi
31     const char *suits[] = { "Hearts", "Diamonds", "Clubs", "Spades" };
32
33     srand(time(NULL)); // randomizza
34
35     fillDeck(deck, faces, suits); // carica il mazzo con le carte
36     shuffle(deck); // metti le carte in ordine casuale
37     deal(deck); // distribuisci tutte le 52 carte
38 }
39
40 // metti le stringhe nelle strutture Card
41 void fillDeck(Card * const deck, const char * faces[],
42     const char * suits[]) {
43     // effettua un'iterazione attraverso deck
44     for (size_t i = 0; i < CARDS; ++i) {
45         deck[i].face = faces[i % FACES];
46         deck[i].suit = suits[i / FACES];
47     }
48 }
49
50 // mescola le carte
51 void shuffle(Card * const deck) {
52     // effettua un'iterazione attraverso deck scambiando a caso le carte
53     for (size_t i = 0; i < CARDS; ++i) {
54         size_t j = rand() % CARDS;
55         Card temp = deck[i];
56         deck[i] = deck[j];
57         deck[j] = temp;
58     }
59 }
60
61 // distribuisci le carte
62 void deal(const Card * const deck) {
63     // effettua un'iterazione attraverso deck
64     for (size_t i = 0; i < CARDS; ++i) {
65         printf("%5s of %-8s%", deck[i].face, deck[i].suit,
66             (i + 1) % 4 ? " " : "\n");
67     }
68 }
```

Three of Hearts	Jack of Clubs	Three of Spades	Six of Diamonds
Five of Hearts	Eight of Spades	Three of Clubs	Deuce of Spades
Jack of Spades	Four of Hearts	Deuce of Hearts	Six of Clubs
Queen of Clubs	Three of Diamonds	Eight of Diamonds	King of Clubs
King of Hearts	Eight of Hearts	Queen of Hearts	Seven of Clubs
Seven of Diamonds	Nine of Spades	Five of Clubs	Eight of Clubs
Six of Hearts	Deuce of Diamonds	Five of Spades	Four of Clubs
Deuce of Clubs	Nine of Hearts	Seven of Hearts	Four of Spades
Ten of Spades	King of Diamonds	Ten of Hearts	Jack of Diamonds
Four of Diamonds	Six of Spades	Five of Diamonds	Ace of Diamonds
Ace of Clubs	Jack of Hearts	Ten of Clubs	Queen of Diamonds
Ace of Hearts	Ten of Diamonds	Nine of Clubs	King of Spades
Ace of Spades	Nine of Diamonds	Seven of Spades	Queen of Spades

Figura 10.2 Programma per mescolare e distribuire le carte con uso di strutture.

La riga 35 chiama la funzione `fillDeck` (righe 41-48) per inizializzare l'array di `Card` in ordine da "Ace" a "King" per ciascun seme. La riga 36 passa l'array di `Card` alla funzione `shuffle` (righe 51-59), che implementa l'algoritmo ad alte prestazioni per mescolare. La funzione `shuffle` prende un array di 52 carte come argomento. La funzione effettua un'iterazione sulle 52 `Card`. Per ogni `Card`, l'algoritmo sceglie un numero a caso tra 0 e 51, poi scambia la carta corrente e la carta selezionata a caso. L'algoritmo effettua 52 scambi in una singola passata lungo l'intero array, e l'array delle carte è mescolato! Questo algoritmo non può subire il fenomeno della posposizione indefinita come l'algoritmo di mescolamento presentato nel Capitolo 7. Le carte sono state scambiate di posto nell'array, pertanto l'algoritmo ad alte prestazioni per la distribuzione della funzione `deal` (righe 62-68) può distribuire le carte mescolate con *una* sola passata dell'array.

Esercizio correlato – Algoritmo di mescolamento Fisher-Yates

Si consiglia di utilizzare un algoritmo di mescolamento imparziale per i giochi di carte reali. Un tale algoritmo garantisce che tutte le possibili sequenze di carte mescolate abbiano le stesse probabilità di verificarsi. L'Esercizio 10.18 chiede di fare ricerche sul popolare algoritmo di mescolamento imparziale Fisher-Yates e di usarlo per reimplementare la funzione `shuffle` della Figura 10.2.

✓ Autovalutazione

- (Codice) Riscrivete il codice seguente in modo da evitare l'istruzione `typedef` separata:

```
struct name {
    const char *first;
    const char *last;
};

typedef struct name Name;
```

Risposta:

```
typedef struct name {
    const char *first;
    const char *last;
} Name;
```

- (Codice) Correggete il codice seguente, che dovrebbe scambiare gli elementi `i` e `j` dell'array di carte (`Card`) `deck`:

```
deck[i] = deck[j];
deck[j] = deck[i];
```

Risposta:

```
Card temp = deck[i];
deck[j] = deck[i];
deck[i] = temp;
```

3. (*Discussione*) Perché l'algoritmo per mescolare le carte che abbiamo presentato nel Capitolo 7 subiva il fenomeno della posposizione indefinita? Perché invece l'algoritmo per mescolare le carte illustrato in questo capitolo non subisce tale problema?

Risposta: Nell'esempio del Capitolo 7 per mescolare e distribuire le carte è stato usato un array 4-per-13 per rappresentare i 4 semi e i 13 valori di un mazzo di carte. L'algoritmo di mescolamento ha usato un ciclo controllato da sentinella per mettere i valori 1-52 (che rappresentano l'ordine di distribuzione) in righe e colonne selezionate a caso. Questo ciclo può essere eseguito indefinitamente nel caso che le celle selezionate a caso contengano già uno di questi valori. L'algoritmo di mescolamento di questo capitolo usa un'iterazione controllata da contatore per effettuare una passata lungo un array unidimensionale. Il ciclo itera una volta per ciascuna carta, poi termina, e quindi non può subire il problema della posposizione indefinita.

10.8 Unioni

Come una struttura, un'unione è un tipo di dati derivato, ma i suoi membri condividono lo stesso spazio di memoria. Per diverse situazioni in un programma, alcune variabili possono non essere rilevanti mentre altre possono esserlo, e così un'unione fa condividere spazio di memoria invece di sprecarlo per variabili che non vengono usate.

I membri di un'unione possono appartenere a qualunque tipo di dati. Il numero di byte utilizzati per memorizzare un'unione deve essere sufficiente almeno a contenere il membro più grande.

Nella maggior parte dei casi, le unioni contengono due o più elementi di tipi differenti. Si può fare riferimento soltanto a un membro alla volta – e di conseguenza a un solo tipo – alla volta. È vostra responsabilità assicurarvi di fare riferimento ai dati con il tipo appropriato. Fare riferimento ai dati correntemente memorizzati con una variabile del tipo sbagliato è un errore logico (il risultato è dipendente dall'implementazione).

Portabilità delle unioni

La quantità di memoria necessaria per memorizzare un'unione è dipendente dall'implementazione. L'operatore `sizeof` restituirà sempre un valore di una grandezza almeno equivalente alla dimensione in byte del membro più grande dell'unione. È possibile che alcune unioni non siano facilmente portabili su altri sistemi. Il fatto che un'unione sia portabile o meno dipende spesso dai requisiti di allineamento nella memoria per i tipi dei membri dell'unione su un dato sistema.

10.8.1 Dichiarazione di unioni

La seguente unione (union) ha due membri, `int x` e `double y`:

```
union number {
    int x;
    double y;
};
```

La definizione di un'unione è normalmente posta in un file di intestazione e inclusa in tutti i file sorgente che usano il tipo `union`. Come con una definizione `struct`, una definizione `union` crea semplicemente un nuovo tipo. Non riserva memoria finché non si utilizza il tipo per creare variabili.

10.8.2 Operazioni consentite sulle unioni

Le operazioni che possono essere eseguite su un'unione (`union`) sono:

- assegnare un'unione a un'altra unione dello stesso tipo;
- accedere all'indirizzo (&) di una variabile di tipo unione;
- accedere ai membri di un'unione usando l'operatore di membro di struttura (.) e l'operatore di puntatore a struttura (->);
- inizializzare a zero l'unione.

Due unioni non possono essere confrontate usando gli operatori `==` e `!=` per le stesse ragioni per cui due strutture non possono essere confrontate.

10.8.3 Inizializzare unioni nelle dichiarazioni

Si può inizializzare un'unione in una dichiarazione con un valore dello stesso tipo del primo membro dell'unione. La definizione `union number` del Paragrafo 10.8.1 ha come suo primo membro un `int`, perciò è possibile inizializzare un oggetto di questo tipo con la seguente istruzione:

```
union number value = {10};
```

Se si inizializza l'oggetto con un tipo `double`, come in

```
union number value = {1.43};
```

C troncherà la parte dopo il punto decimale del valore dell'inizializzatore (alcuni compilatori forniranno un messaggio di avvertimento a tale riguardo).

10.8.4 Illustrazione delle unioni

Il programma della Figura 10.3 stampa una variabile di tipo `union number` chiamata `value` (riga 12) sia come un `int` che come un `double`. L'output di questo programma è dipendente dall'implementazione.

```
1 // fig10_03.c
2 // Stampa del valore di un'unione con entrambi i tipi di dati dei membri
3 #include <stdio.h>
4
5 // definizione dell'unione number
6 union number {
7     int x;
8     double y;
9 };
10
11 int main(void) {
12     union number value; // definisci la variabile di tipo union
13
14     value.x = 100; // inserisci un int nell'unione
15     puts("Put 100 in the int member and print both members:");
16     printf("int: %d\ndouble: %.2f\n\n", value.x, value.y);
17
18     value.y = 100.0; // inserisci un double nella stessa unione
19     puts("Put 100.0 in the double member and print both members:");
20     printf("int: %d\ndouble: %.2f\n\n", value.x, value.y);
21 }
```

Microsoft Visual Studio

```
Put 100 in the int member and print both members:
int: 100
double: -92559592117433135502616407313071917486139351398276445610442752.00

Put 100.0 in the double member and print both members:
int: 0
double: 100.00
```

GNU GCC e Apple Xcode

```
Put 100 in the int member and print both members:
int: 100
double: 0.00

Put 100.0 in the double member and print both members:
int: 0
double: 100.00
```

Figura 10.3 Stampa del valore di un'unione con entrambi i tipi di dati dei membri.

✓ Autovalutazione

1. (*Discussione*) Come le strutture, le unioni sono tipi di dati derivati. In cosa differisce un'unione da una struttura?

Risposta: Un'unione usa la stessa memoria per tutti i suoi membri. Solo un membro alla volta può essere memorizzato in un'unione. Si deve tenere traccia di quale membro è correntemente memorizzato.

2. (*Vero/Falso*) La definizione seguente di union indica che number è del tipo union con membri int x e double y:

```
union number {
    int x;
    double y;
};
```

Su una macchina con int di quattro byte e double di otto byte, il compilatore deve riservare almeno 12 byte per una variabile di questo tipo union.

Risposta: Falso. Solo uno di questi membri alla volta è attivo, perciò l'*unione* deve riservare soltanto lo spazio di memoria sufficiente a contenere il suo membro più grande: in questo caso, otto byte.

10.9 Operatori bit a bit

I computer rappresentano tutti i dati internamente come sequenze di bit. Ogni bit può assumere il valore 0 o il valore 1. Sulla maggior parte dei sistemi, una sequenza di 8 bit forma un byte, la tipica unità di memoria per una variabile di tipo char. Gli operatori bit a bit sono usati per manipolare i bit di operandi interi, sia signed che unsigned, sebbene di solito vengano usati gli interi senza segno. Le manipolazioni dei dati bit a bit sono dipendenti dalla macchina. La tabella seguente presenta un riepilogo degli operatori bit a bit.

Operatore	Descrizione
&	AND bit a bit Confronta i suoi due operandi bit per bit. I bit nel risultato sono impostati a 1 se i bit corrispondenti nei due operandi sono <i>entrambi</i> 1.
	OR inclusivo bit a bit Confronta i suoi due operandi bit per bit. I bit nel risultato sono impostati a 1 se <i>almeno uno</i> dei bit corrispondenti nei due operandi è 1.
^	OR esclusivo bit a bit (conosciuto anche come XOR bit a bit) Confronta i suoi due operandi bit per bit. I bit nel risultato sono impostati a 1 se i bit corrispondenti nei due operandi sono differenti.
<<	spostamento a sinistra Sposta a sinistra i bit del primo operando del numero di bit specificato dal secondo operando; i bit da destra vengono riempiti con 0.
>>	spostamento a destra Sposta a destra i bit del primo operando del numero di bit specificato dal secondo operando; il metodo di riempimento da sinistra è dipendente dalla macchina quando l'operando sinistro è negativo.
-	complemento Tutti i bit 0 sono impostati a 1 e tutti i bit 1 sono impostati a 0 ("alternanza di bit").

Gli esempi che seguono presentano un'analisi dettagliata di ogni operatore bit a bit. Gli esempi mostrano le rappresentazioni binarie degli operandi interi.

10.9.1 Stampa di un intero senza segno nella sua rappresentazione in bit

Quando si usano gli operatori bit a bit, è utile stampare i valori in binario¹ per mostrare gli effetti esatti di questi operatori. Il programma della Figura 10.4 stampa un unsigned int nella sua rappresentazione binaria in gruppi di otto bit ciascuno per facilitarne la leggibilità. Tutti i compilatori che abbiamo usato per testare questi esempi memorizzano degli unsigned int in 4 byte (32 bit) di memoria.

```

1 // fig10_04.c
2 // Stampa di un int senza segno nella sua rappresentazione in bit
3 #include <stdio.h>
4
5 void displayBits(unsigned int value); // prototipo
6
7 int main(void) {
8     unsigned int x = 0; // variabile per memorizzare l'input dell'utente
9
10    printf("%s", "Enter a nonnegative int: ");
11    scanf("%u", &x);
12    displayBits(x);
13 }
14
15 // stampa i bit di un valore int senza segno
16 void displayBits(unsigned int value) {
17     // definisci displayMask ed effettua uno spostamento a sinistra di 31 bit
18     unsigned int displayMask = 1 << 31;
19
20     printf("%10u = ", value);
21
22     // effettua un'iterazione attraverso tutti i bit
23     for (unsigned int c = 1; c <= 32; ++c) {

```

1. Consultate l'Appendice E (disponibile sulla piattaforma MyLab) per una descrizione dettagliata del sistema numerico binario (in base 2).

```

24     putchar(value & displayMask ? '1' : '0');
25     value <<= 1; // sposta value a sinistra di 1 bit
26
27     if (c % 8 == 0) { // stampa uno spazio dopo 8 bit
28         putchar(' ');
29     }
30 }
31
32     putchar('\n');
33 }
```

Enter a nonnegative int: 65000
 65000 = 00000000 00000000 11111101 11101000

Figura 10.4 Stampa di un `int` senza segno nella sua rappresentazione in bit.

Stampare i bit di un intero

La funzione `displayBits` (righe 16-33) utilizza l'operatore bit a bit AND per combinare la variabile `value` con la variabile `displayMask` (riga 24). Spesso, l'operatore bit a bit AND è usato con un operando chiamato **maschera**, un valore intero con bit specifici impostati a 1. Le maschere servono per nascondere alcuni bit in un valore mentre vengono selezionati altri bit. Nella funzione `displayBits`, la riga 18 assegna alla variabile maschera `displayMask` il valore

`1 << 31` (10000000 00000000 00000000 00000000)

L'operatore di spostamento a sinistra sposta il valore 1 dal bit di ordine più basso (più a destra) al bit di ordine più alto (più a sinistra) in `displayMask` ed effettua il riempimento con bit 0 da destra. La riga 24

`putchar(value & displayMask ? '1' : '0');`

determina se si deve stampare un 1 o uno 0 per il bit corrente più a sinistra della variabile `value`. Quando `value` e `displayMask` vengono combinati usando `&`, tutti i bit eccetto il bit di ordine più alto nella variabile `value` sono “mascherati” (nascosti), perché ogni bit “in AND” con 0 dà 0. Se il bit più a sinistra è 1, `value & displayMask` dà un valore diverso da zero (vero) e la riga 24 stampa 1; altrimenti, stampa 0. La riga 25 sposta a sinistra di 1 bit la variabile `value` con l'espressione `value <<= 1`. La funzione `displayBits` ripete questi passi per ogni bit in `value`. Usare l'operatore AND logico (`&&`) al posto dell'operatore AND (`&`) bit a bit – e viceversa – è un errore logico. La tabella che segue riepiloga i risultati della combinazione di due bit con l'operatore bit a bit AND.

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

10.9.2 Rendere la funzione `displayBits` più generica e portabile

Nella riga 18 della Figura 10.4, abbiamo codificato l'intero 31 per indicare che il valore 1 andava spostato al bit più a sinistra nella variabile `displayMask`. In modo simile, nella riga 23, abbiamo codificato l'intero 32 per indicare che il ciclo deve essere iterato 32 volte, una volta per ogni bit nella variabile `value`. Abbiamo supposto che gli `unsigned int` fossero sempre memorizzati in 32 bit (quattro byte) di memoria. Molti dei computer odierni più diffusi usano architetture hardware con parole di 32 o 64 bit. Come programmatore in C, dovrete

lavorare su tante diverse architetture hardware e qualche volta gli `unsigned int` saranno memorizzati con un numero di bit più piccolo o più grande.

 Possiamo rendere il programma della Figura 10.4 più generico e più portabile sostituendo gli interi 31 (riga 18) e 32 (riga 23) con espressioni che calcolino questi interi, in base alla dimensione di un `unsigned int` per la piattaforma sulla quale viene eseguito il programma. La costante simbolica `CHAR_BIT` (definita in `<limits.h>`) rappresenta il numero di bit in un byte (normalmente 8). Ricordatevi che `sizeof` determina il numero di byte usati per memorizzare un oggetto o un tipo. L'espressione `sizeof(unsigned int)` assume il valore 4 per `unsigned int` a 32 bit e 8 per `unsigned int` a 64 bit. Potete sostituire 31 con

```
CHAR_BIT * sizeof(unsigned int) - 1
```

e sostituire 32 con

```
CHAR_BIT * sizeof(unsigned int)
```

Per `unsigned int` a 32 bit, queste espressioni assumono il valore 31 e 32. Per `unsigned int` a 64 bit, assumono il valore 63 e 64.

10.9.3 Uso degli operatori bit a bit AND, OR inclusivo, OR esclusivo e complemento

Il programma della Figura 10.5 illustra l'uso degli operatori bit a bit AND, OR inclusivo, OR esclusivo e complemento. Il programma usa la funzione `displayBits` (righe 45-62) per stampare i valori `unsigned int`.

```

1 // fig10_05.c
2 // Uso degli operatori bit a bit AND, OR inclusivo,
3 // OR esclusivo e complemento
4 #include <stdio.h>
5
6 void displayBits(unsigned int value); // prototipo
7
8 int main(void) {
9     // illustra l'AND (&) bit a bit
10    unsigned int number1 = 65535;
11    unsigned int mask = 1;
12    puts("The result of combining the following");
13    displayBits(number1);
14    displayBits(mask);
15    puts("using the bitwise AND operator & is");
16    displayBits(number1 & mask);
17
18    // illustra l'OR inclusivo (|) bit a bit
19    number1 = 15;
20    unsigned int setBits = 241;
21    puts("\nThe result of combining the following");
22    displayBits(number1);
23    displayBits(setBits);
24    puts("using the bitwise inclusive OR operator | is");
25    displayBits(number1 | setBits);
26
27    // illustra l'OR esclusivo (^) bit a bit
28    number1 = 139;
29    unsigned int number2 = 199;
30    puts("\nThe result of combining the following");
31    displayBits(number1);
32    displayBits(number2);
33    puts("using the bitwise exclusive OR operator ^ is");
```

```

34     displayBits(number1 ^ number2);
35
36     // illustra il complemento (~) bit a bit
37     number1 = 21845;
38     puts("\nThe one's complement of");
39     displayBits(number1);
40     puts("is");
41     displayBits(~number1);
42 }
43
44 // stampa i bit di un valore unsigned int
45 void displayBits(unsigned int value) {
46     // dichiara displayMask ed effettua uno spostamento a sinistra di 31 bit
47     unsigned int displayMask = 1 << 31;
48
49     printf("%10u = ", value);
50
51     // effettua un'iterazione attraverso tutti i bit
52     for (unsigned int c = 1; c <= 32; ++c) {
53         putchar(value & displayMask ? '1' : '0');
54         value <<= 1; // sposta value a sinistra di 1 bit
55
56         if (c % 8 == 0) { // stampa uno spazio dopo 8 bit
57             putchar(' ');
58         }
59     }
60
61     putchar('\n');
62 }
```

```

The result of combining the following
65535 = 00000000 00000000 11111111 11111111
      1 = 00000000 00000000 00000000 00000001
using the bitwise AND operator & is
      1 = 00000000 00000000 00000000 00000001

The result of combining the following
      15 = 00000000 00000000 00000000 00001111
      241 = 00000000 00000000 00000000 11110001
using the bitwise inclusive OR operator | is
      255 = 00000000 00000000 00000000 11111111

The result of combining the following
      139 = 00000000 00000000 00000000 10001011
      199 = 00000000 00000000 00000000 11000111
using the bitwise exclusive OR operator ^ is
      76 = 00000000 00000000 00000000 01001100

The one's complement of
      21845 = 00000000 00000000 01010101 01010101
is
      4294945450 = 11111111 11111111 10101010 10101010
```

Figura 10.5 Uso degli operatori bit a bit AND, OR inclusivo, OR esclusivo e complemento.

Operatore bit a bit AND (&)

La riga 10 assegna il valore 65535

```
00000000 00000000 11111111 11111111
```

alla variabile intera number1, e la riga 11 assegna il valore 1

```
00000000 00000000 00000000 00000001
```

alla variabile mask. Quando si combinano number1 e mask usando l'operatore bit a bit AND (&) nell'espressione number1 & mask (riga 16), il risultato è

```
00000000 00000000 00000000 00000001
```

Tutti i bit eccetto il bit di ordine più basso in number1 sono "mascherati" (nascosti) da un'operazione AND con la variabile mask.

Operatore bit a bit OR inclusivo (|)

L'operatore bit a bit OR inclusivo imposta specifici bit a 1 in un operando. La riga 19 assegna 15

```
00000000 00000000 00000000 00001111
```

alla variabile number1 e la riga 20 assegna 241

```
00000000 00000000 00000000 11110001
```

alla variabile setBits. Quando si combinano number1 e setBits usando l'operatore bit a bit OR inclusivo nell'espressione number1 | setBits (riga 25), il risultato è 255

```
00000000 00000000 00000000 11111111
```

La tabella seguente riepiloga i risultati della combinazione di due bit con l'operatore bit a bit OR inclusivo.

Bit 1	Bit 2	Bit 1 Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

Operatore bit a bit OR esclusivo (^)

L'operatore bit a bit OR esclusivo (^) imposta ogni bit nel risultato a 1 se esattamente uno soltanto dei bit corrispondenti nei suoi due operandi è 1. La riga 28 assegna a number1 il valore 139

```
00000000 00000000 00000000 10001011
```

e la riga 29 assegna a number2 il valore 199

```
00000000 00000000 00000000 11000111
```

Quando si combinano queste variabili con l'operatore bit a bit OR esclusivo nell'espressione number1 ^ number2 (riga 34), il risultato è

```
00000000 00000000 00000000 01001100
```

La tabella seguente riepiloga i risultati della combinazione di due bit con l'operatore bit a bit OR esclusivo.

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

Operatore bit a bit complemento (\sim)

L'operatore bit a bit complemento (\sim) imposta tutti i bit che sono 1 nel suo operando a 0 nel risultato e imposta tutti i bit che sono 0 a 1, ossia “calcola il **complemento a uno** del valore”. La riga 37 assegna a `number1` il valore 21845

```
00000000 00000000 01010101 01010101
```

L'espressione `\sim number1` (riga 41) inverte tutti i bit dando come risultato

```
11111111 11111111 10101010 10101010
```

10.9.4 Uso degli operatori bit a bit di spostamento a sinistra e a destra

Il programma della Figura 10.6 illustra l'operatore di spostamento a sinistra (`<<`) e l'operatore di spostamento a destra (`>>`). Anche in questo caso, viene usata la funzione `displayBits` per stampare i valori `unsigned int`.

```

1 // fig10_06.c
2 // Uso degli operatori di spostamento bit a bit
3 #include <stdio.h>
4
5 void displayBits(unsigned int value); // prototipo
6
7 int main(void) {
8     unsigned int number1 = 960; // inizializza number1
9
10    // illustra lo spostamento a sinistra bit a bit
11    puts("\nThe result of left shifting");
12    displayBits(number1);
13    puts("8 bit positions using the left shift operator << is");
14    displayBits(number1 << 8);
15
16    // illustra lo spostamento a destra bit a bit
17    puts("\nThe result of right shifting");
18    displayBits(number1);
19    puts("8 bit positions using the right shift operator >> is");
20    displayBits(number1 >> 8);
21 }
22
23 // stampa i bit di un valore unsigned int
24 void displayBits(unsigned int value) {
25     // dichiara displayMask ed effettua uno spostamento a sinistra di 31 bit
26     unsigned int displayMask = 1 << 31;
27
28     printf("%10u = ", value);

```

```

29
30     // effettua un'iterazione attraverso tutti i bit
31     for (unsigned int c = 1; c <= 32; ++c) {
32         putchar(value & displayMask ? '1' : '0');
33         value <<= 1; // sposta value a sinistra di 1 bit
34
35         if (c % 8 == 0) { // stampa uno spazio dopo 8 bit
36             putchar(' ');
37         }
38     }
39
40     putchar('\n');
41 }
```

```

The result of left shifting
960 = 00000000 00000000 00000011 11000000
8 bit positions using the left shift operator << is
245760 = 00000000 00000011 11000000 00000000

The result of right shifting
960 = 00000000 00000000 00000011 11000000
8 bit positions using the right shift operator >> is
3 = 00000000 00000000 00000000 00000011
```

Figura 10.6 Uso degli operatori di spostamento bit a bit.

Operatore di spostamento a sinistra (<<)

L'operatore di spostamento a sinistra (<<) sposta a sinistra i bit del suo operando sinistro del numero di bit specificato nel suo operando destro. I bit vuoti a destra sono sostituiti con zeri. I bit spostati a sinistra vanno perduti. La riga 8 assegna alla variabile `number1` il valore 960

```
00000000 00000000 00000011 11000000
```

Lo spostamento a sinistra di otto byte di `number1` con l'espressione `number1 << 8` (riga 14) risulta nel valore 245760

```
00000000 00000011 11000000 00000000
```

Operatore di spostamento a destra (>>)

L'operatore di spostamento a destra (>>) sposta a destra i bit del suo operando sinistro del numero di bit specificato nel suo operando destro. Lo spostamento a destra di un `unsigned int` fa sì che vengano sostituiti con zeri tutti i bit rimasti vuoti a sinistra. I bit spostati a destra vanno perduti. Il risultato dello spostamento a destra di `number1` con l'espressione `number1 >> 8` (riga 20) è 3

```
00000000 00000000 00000000 00000011
```



Il risultato dello spostamento a destra o a sinistra di un valore è indefinito se l'operando destro è negativo o se l'operando destro è più grande del numero di bit nell'operando sinistro. Il risultato dello spostamento a destra di un numero negativo è definito in base all'implementazione.

10.9.5 Operatori di assegnazione bit a bit

Ogni operatore bit a bit binario ha un operatore di assegnazione corrispondente. Questi **operatori di assegnazione bit a bit** sono elencati nella tabella seguente.

Operatori di assegnazione bit a bit	
&=	Operatore di assegnazione AND bit a bit.
 =	Operatore di assegnazione OR inclusivo bit a bit.
^=	Operatore di assegnazione OR esclusivo bit a bit.
<<=	Operatore di assegnazione di spostamento a sinistra.
>>=	Operatore di assegnazione di spostamento a destra.

La tabella seguente mostra in ordine decrescente la precedenza e l'associatività dei vari operatori introdotti fino a questo punto nel testo.

Operatore	Associatività	Tipo
() [] . -> ++(postfisso) --(postfix)	da sinistra a destra	con precedenza più alta
+ - ++ -- ! & * ~ sizeof (tipo)	da destra a sinistra	unario
* / %	da sinistra a destra	moltiplicativo
+ -	da sinistra a destra	additivo
<< >>	da sinistra a destra	di spostamento
< <= > >=	da sinistra a destra	relazionale
== !=	da sinistra a destra	di uguaglianza
&	da sinistra a destra	AND bit a bit
^	da sinistra a destra	XOR bit a bit
 	da sinistra a destra	OR bit a bit
&&	da sinistra a destra	AND logico
 	da sinistra a destra	OR logico
? :	da sinistra a destra	condizionale
- += -= *= /= %= &= = ^= <<= >>=	da sinistra a destra	di assegnazione
,	da sinistra a destra	virgola

✓ Autovalutazione

1. (*Completere*) Spesso, l'operatore bit a bit AND è usato con un operando chiamato _____, che è un valore intero con bit specifici impostati a 1. Questo operando è usato per nascondere alcuni bit in un valore mentre ne vengono selezionati altri.

Risposta: maschera.

2. (*Vero/Falso*) L'operatore bit a bit OR esclusivo impone ogni bit nel risultato a 1 se il bit corrispondente nell'uno o nell'altro operando o in entrambi è 1.

Risposta: Falso. In realtà, l'operatore sopra descritto è l'operatore bit a bit OR inclusivo. L'operatore bit a bit OR esclusivo impone ogni bit nel risultato a 1 se i bit corrispondenti di ogni operando sono differenti.

3. (*Completere*) L'operatore bit a bit complemento imposta a 1 tutti i bit che sono 0 nel suo operando nel risultato e imposta a 0 tutti i bit che sono 1. Questa operazione viene spesso chiamata _____ di bit.

Risposta: alternanza.

4. (*Vero/Falso*) Poiché le manipolazioni bit a bit sono dipendenti dalla macchina, è possibile che i programmi che le includono non funzionino correttamente o funzionino in modo differente sui vari sistemi.

Risposta: Vero.

10.10 Campi di bit

È possibile specificare il numero di bit in cui memorizzare un membro intero `unsigned` o `signed` di una struttura o di un'unione. Questi costrutti, chiamati **campi di bit**, permettono un migliore utilizzo della memoria, memorizzando i dati nel minimo numero di bit necessario. I membri di un campo di bit solitamente sono dichiarati come `int` o `unsigned int`.

10.10.1 Definire campi di bit

La seguente struct `bitCard`

```
struct bitCard {
    unsigned int face : 4;
    unsigned int suit : 2;
    unsigned int color : 1;
};
```

contiene tre campi di bit `unsigned int` -- `face`, `suit` e `color` -- usati per rappresentare una carta di un mazzo di 52 carte. Un campo di bit viene dichiarato facendo seguire il nome di un membro intero `unsigned` o `signed` da due punti (:) e da una costante intera che rappresenta la **larghezza** del campo (cioè il numero di bit in cui il membro è memorizzato). La costante che rappresenta la larghezza del campo deve essere un intero tra 0 e il numero totale di bit usati per memorizzare un `int` sul vostro sistema (incluso). I nostri esempi sono stati testati su un computer con interi a 4 byte (32 bit).

La definizione di struttura precedente indica che i membri `face`, `suit` e `color` sono memorizzati rispettivamente in 4 bit, 2 bit e 1 bit. Il numero di bit si basa sull'intervallo di valori desiderato per ogni membro:

- `face` memorizza valori da 0 (Ace) a 12 (King); 4 bit possono memorizzare valori nell'intervallo 0-15;
- `suit` memorizza valori da 0 a 3 (0 = Hearts, 1 = Diamonds, 2 = Clubs, 3 = Spades); 2 bit possono memorizzare valori nell'intervallo 0-3;
- `color` memorizza 0 (Red) o 1 (Black); 1 bit può memorizzare 0 o 1.

10.10.2 Usare campi di bit per rappresentare i membri `face`, `suit` e `color` di una carta

Il programma della Figura 10.7 crea l'array `deck` contenente 52 strutture `struct bitCard` nella riga 19. La funzione `fillDeck` (righe 30-37) inserisce le 52 carte nell'array `deck`, e la funzione `deal` (righe 41-49) stampa le 52 carte. Notate che ai membri campi di bit delle strutture si accede esattamente come a un qualsiasi altro membro della struttura.

```
1 // fig10_07.c
2 // Rappresentazione di carte con campi di bit in una struttura
3 #include <stdio.h>
4 #define CARDS 52
5
6 // definizione della struttura bitCard con campi di bit
7 struct bitCard {
8     unsigned int face : 4; // 4 bit; 0-15
9     unsigned int suit : 2; // 2 bit; 0-3
10    unsigned int color : 1; // 1 bit; 0-1
11};
12
13 typedef struct bitCard Card; // nuovo nome di tipo per la struttura bitCard
14
15 void fillDeck(Card * const deck); // prototipo
16 void deal(const Card * const deck); // prototipo
17
18 int main(void) {
19     Card deck[CARDS]; // crea un array di Card
20
21     fillDeck(deck);
22
23     puts("Card values 0-12 correspond to Ace through King");
24     puts("Suit values 0-3 correspond to Hearts, Diamonds, Clubs and Spades");
25     puts("Color values 0-1 correspond to red and black\n");
26     deal(deck);
27 }
28
29 // inizializza l'array di Card
30 void fillDeck(Card * const deck) {
31     // effettua un'iterazione attraverso deck
32     for (size_t i = 0; i < CARDS; ++i) {
33         deck[i].face = i % (CARDS / 4);
34         deck[i].suit = i / (CARDS / 4);
35         deck[i].color = i / (CARDS / 2);
36     }
37 }
38
39 // stampa le carte nel formato a due colonne; le carte 0-25 indicizzate con
40 // k1 (colonna 1); le carte 26-51 indicizzate con k2 (colonna 2)
41 void deal(const Card * const deck) {
42     // effettua l'iterazione attraverso deck
43     for (size_t k1 = 0, k2 = k1 + 26; k1 < CARDS / 2; ++k1, ++k2) {
44         printf("Card:%3d Suit:%2d Color:%2d  ",
45             deck[k1].face, deck[k1].suit, deck[k1].color);
46         printf("Card:%3d Suit:%2d Color:%2d\n",
47             deck[k2].face, deck[k2].suit, deck[k2].color);
48     }
49 }
```

```

Card values 0-12 correspond to Ace through King
Suit values 0-3 correspond to Hearts, Diamonds, Clubs and Spades
Color values 0-1 correspond to red and black

Card: 0 Suit: 0 Color: 0   Card: 0 Suit: 2 Color: 1
Card: 1 Suit: 0 Color: 0   Card: 1 Suit: 2 Color: 1
Card: 2 Suit: 0 Color: 0   Card: 2 Suit: 2 Color: 1
Card: 3 Suit: 0 Color: 0   Card: 3 Suit: 2 Color: 1
Card: 4 Suit: 0 Color: 0   Card: 4 Suit: 2 Color: 1
Card: 5 Suit: 0 Color: 0   Card: 5 Suit: 2 Color: 1
Card: 6 Suit: 0 Color: 0   Card: 6 Suit: 2 Color: 1
Card: 7 Suit: 0 Color: 0   Card: 7 Suit: 2 Color: 1
Card: 8 Suit: 0 Color: 0   Card: 8 Suit: 2 Color: 1
Card: 9 Suit: 0 Color: 0   Card: 9 Suit: 2 Color: 1
Card: 10 Suit: 0 Color: 0  Card: 10 Suit: 2 Color: 1
Card: 11 Suit: 0 Color: 0  Card: 11 Suit: 2 Color: 1
Card: 12 Suit: 0 Color: 0  Card: 12 Suit: 2 Color: 1
Card: 0 Suit: 1 Color: 0   Card: 0 Suit: 3 Color: 1
Card: 1 Suit: 1 Color: 0   Card: 1 Suit: 3 Color: 1
Card: 2 Suit: 1 Color: 0   Card: 2 Suit: 3 Color: 1
Card: 3 Suit: 1 Color: 0   Card: 3 Suit: 3 Color: 1
Card: 4 Suit: 1 Color: 0   Card: 4 Suit: 3 Color: 1
Card: 5 Suit: 1 Color: 0   Card: 5 Suit: 3 Color: 1
Card: 6 Suit: 1 Color: 0   Card: 6 Suit: 3 Color: 1
Card: 7 Suit: 1 Color: 0   Card: 7 Suit: 3 Color: 1
Card: 8 Suit: 1 Color: 0   Card: 8 Suit: 3 Color: 1
Card: 9 Suit: 1 Color: 0   Card: 9 Suit: 3 Color: 1
Card: 10 Suit: 1 Color: 0  Card: 10 Suit: 3 Color: 1
Card: 11 Suit: 1 Color: 0  Card: 11 Suit: 3 Color: 1
Card: 12 Suit: 1 Color: 0  Card: 12 Suit: 3 Color: 1

```

Figura 10.7 Rappresentazione di carte con campi di bit in una struttura.

- ❖ I campi di bit riducono la quantità di memoria di cui ha bisogno un programma, ma sono dipendenti dalla macchina. Sebbene i campi di bit facciano risparmiare spazio di memoria, usarli può far sì che il compilatore generi un codice in linguaggio macchina che viene eseguito più lentamente. Ciò accade perché sono necessarie ulteriori operazioni in linguaggio macchina per accedere soltanto a porzioni di un'unità di memoria indirizzabile. Questo è uno dei molti esempi di compromesso spazio-tempo che ricorrono nell'informatica.
- ❖ I campi di bit non hanno indirizzi, pertanto tentare di accedere all'indirizzo di un campo di bit con l'operatore & è un errore. Anche usare l'operatore sizeof con un campo di bit è un errore.

10.10.3 Campi di bit anonimi

Un campo di bit anonimo viene usato per occupare lo spazio libero in una struttura. Per esempio, la definizione

```

struct example {
    unsigned int a : 13;
    unsigned int : 19;
    unsigned int b : 4;
};

```

usa un campo anonimo di 19 bit per occupare lo spazio libero della struttura: non è possibile memorizzare niente in quei 19 bit. Il membro `b` (sul nostro computer con parole di 4 byte) è memorizzato in un'altra unità di memoria.

Un **campo di bit anonimo con larghezza zero** viene usato per allineare il successivo campo di bit su un nuovo confine di unità di memoria. Per esempio, la definizione di struttura

```
struct example {
    unsigned int a : 13;
    unsigned int : 0;
    unsigned int : 4;
};
```

usa un campo anonimo di 0 bit per tralasciare i restanti bit (tanti quanti ve ne sono) dell'unità di memoria in cui è memorizzato `a` e per allineare `b` sul successivo confine di unità di memoria.

✓ Autovalutazione

1. (*Completare*) La definizione di struttura

```
struct example {
    unsigned int a : 13;
    unsigned int : 19;
    unsigned int b : 4;
};
```

usa un campo anonimo di 19 bit per _____: non è possibile memorizzare niente in quei 19 bit.

Risposta: occupare lo spazio libero della struttura.

2. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) relative alla struttura `struct bitCard` del Paragrafo 10.10.1 è *falsa*?

- Un campo di bit viene dichiarato facendo seguire il nome di un membro intero `unsigned` o `signed` da due punti (`:`) e da una costante intera che rappresenta la larghezza del campo in bit.
- La definizione di `struct bitCard` indica che il membro `face` è memorizzato in 4 bit, il membro `suit` è memorizzato in 2 bit, e il membro `color` è memorizzato in 1 bit.
- Il numero di bit in un campo di bit si basa sull'intervallo di valori desiderato per ogni membro della struttura.
- Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

10.11 Costanti di enumerazione

Nel Paragrafo 5.11 è stata introdotta la parola chiave `enum` per definire un insieme di **costanti di enumerazione** intere rappresentate da identificatori. I valori in un `enum` partono da 0, a meno che non sia specificato diversamente, e sono incrementati nell'ordine di 1. Per esempio, l'enumerazione

```
enum months {
    JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
};
```

crea il nuovo tipo `enum months`, in cui gli identificatori sono impostati con gli interi da 0 a 11. Per numerare i mesi da 1 a 12, usate:

```
enum months {
    JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
};
```

che imposta esplicitamente `JAN` a 1. I valori restanti sono incrementati a partire da 1, dando come risultato i valori da 1 a 12.

 Gli identificatori in qualsiasi enumerazione accessibile in un dato campo d'azione devono essere unici. Il valore di ogni costante di enumerazione può essere impostato esplicitamente nella definizione assegnando un valore all'identificatore. Più membri di un'enumerazione possono avere lo stesso valore costante. Assegnare un valore a una costante di enumerazione dopo che è stata definita è un errore di sintassi. Usate solo lettere maiuscole nei nomi delle costanti di enumerazione per farle risaltare in un programma e per ricordarvi che le costanti di enumerazione non sono variabili.

Nel programma della Figura 10.8 viene usata la variabile di tipo enumerazione month in un'istruzione for per stampare i mesi dell'anno dall'array monthName. In questo esempio abbiamo impostato monthName[0] con la stringa vuota "" e l'abbiamo ignorato.

```

1 // fig10_08.c
2 // Uso di un'enumerazione
3 #include <stdio.h>
4
5 // le costanti di enumerazione rappresentano i mesi dell'anno
6 enum months {
7     JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
8 };
9
10 int main(void) {
11     // inizializza un array di puntatori
12     const char *monthName[] = { "", "January", "February", "March",
13         "April", "May", "June", "July", "August", "September", "October",
14         "November", "December" };
15
16     // effettua un'iterazione attraverso months
17     for (enum months month = JAN; month <= DEC; ++month) {
18         printf("%2d%11s\n", month, monthName[month]);
19     }
20 }
```

```

1 January
2 February
3 March
4 April
5 May
6 June
7 July
8 August
9 September
10 October
11 November
12 December
```

Figura 10.8 Uso di un'enumerazione.

✓ Autovalutazione

1. (*Codice*) L'enumerazione seguente crea un nuovo tipo, enum days, in cui gli identificatori sono impostati con gli interi da 0 a 6:

```
enum days {
    MON, TUE, WED, THU, FRI, SAT, SUN
};
```

Riscrivete questa enumerazione per numerare i giorni da 1 a 7.

Risposta:

```
enum days {
    MON = 1, TUE, WED, THU, FRI, SAT, SUN
};
```

2. (*Vero/Falso*) Più costanti di enumerazione nello stesso campo d'azione possono avere lo stesso identificatore.

Risposta: *Falso*. In realtà, gli identificatori in qualsiasi enumerazione accessibile nello stesso campo d'azione devono essere unici. Più membri di un'enumerazione possono avere lo stesso valore costante.

10.12 Strutture e unioni anonime

Strutture e unioni anonime possono essere annidate in strutture e unioni con nome. I membri di una struttura o di una unione anonima annidata sono considerati membri della struttura o unione che la racchiude. Essi sono accessibili direttamente tramite un oggetto del tipo che racchiude. Per esempio, considerate la seguente dichiarazione di una struct:

```
struct myStruct {
    int member1;
    int member2;

    struct { // struttura anonima
        int nestedMember1;
        int nestedMember2;
    }; // fine della struttura annidata
}; // fine della struttura esterna
```

Per una variabile struct myStruct chiamata object, potete accedere ai membri come

```
object.member1;
object.member2;
object.nestedMember1;
object.nestedMember2;
```

✓ Autovalutazione

1. (*Vero/Falso*) I membri di una struttura o di una unione anonima annidata sono considerati membri della struttura o unione che la racchiude. Essi sono accessibili direttamente tramite un oggetto del tipo che racchiude.

Risposta: *Vero*.



10.13 Programmazione sicura in C

Gli argomenti di questo capitolo riguardano varie linee guida e regole del CERT. Per maggiori informazioni visitate il sito <https://wiki.sei.cmu.edu/>.

Linee guida del CERT per le strutture

Come abbiamo visto nel Paragrafo 10.2.4, i requisiti di allineamento ai confini delle unità di memoria per i membri di tipi struct possono produrre byte extra contenenti dati indefiniti per ogni variabile struct che si crea. Ognuna delle linee guida seguenti riguarda questo problema.

- EXP03-C: a causa dei requisiti di allineamento ai confini, la dimensione di una variabile struct *non* è necessariamente la somma delle dimensioni dei suoi membri. Usate sempre sizeof per determinare il numero di byte in una variabile struct. Useremo questa tecnica per manipolare record di lunghezza fissa scritti su file e letti da file (Capitolo 11) e per creare strutture dinamiche di dati (Capitolo 12).

- EXP04-C: come abbiamo visto nel Paragrafo 10.2.4, le variabili `struct` non possono essere confrontate per egualanza o inegualanza, perché potrebbero contenere byte di dati indefiniti. Di conseguenza, dovete confrontare i loro membri individuali.
- DCL39-C: in una variabile `struct`, i byte extra indefiniti potrebbero contenere dati con requisiti di sicurezza (rimasti dall'uso precedente di quelle locazioni di memoria) che non devono essere accessibili. Questa linea guida del CERT esamina i meccanismi specifici del compilatore per impacchettare i dati al fine di eliminare tali byte extra.

Linea guida del CERT per `typedef`

- DCL05-C: le dichiarazioni complesse di tipo, come quelle per i puntatori a funzioni, possono essere difficili da leggere. Dovreste usare `typedef` per creare nomi di tipo autodocumentanti che rendano più leggibili i vostri programmi.

Linee guida del CERT per la manipolazione di bit

- INT02-C: come conseguenza delle regole di promozione per gli interi (esaminate nel Paragrafo 5.6), l'esecuzione di operazioni bit a bit su tipi interi più piccoli di `int` può portare a risultati inattesi. Sono necessari cast esplicativi per assicurare risultati corretti.
- INT13-C: alcune operazioni bit a bit su tipi interi con segno sono definite in base all'implementazione. Ciò significa che le operazioni possono avere risultati differenti a seconda del compilatore C usato. Per questa ragione, con gli operatori bit a bit vanno usati tipi interi senza segno.
- EXP46-C: gli operatori logici `&&` e `||` sono frequentemente confusi, rispettivamente, con gli operatori bit a bit `&` e `|`. L'uso di `&` e `|` nella condizione di un'espressione condizionale (`? :`) può portare a un comportamento inaspettato, perché gli operatori `&` e `|` non usano la valutazione di tipo corto circuito.

Linea guida del CERT per `enum`

- INT09-C: permettere a più costanti di enumerazione di avere lo stesso valore può generare errori logici difficili da trovare. Nella maggior parte dei casi, le costanti di enumerazione di un `enum` dovrebbero avere valori unici in modo da evitare tali errori logici.

✓ Autovalutazione

1. (*Completare*) Le variabili `struct` non possono essere confrontate per egualanza o inegualanza, perché potrebbero contenere byte di dati indefiniti. Invece, dovete _____.

Risposta: confrontare i loro singoli membri.

2. (*Vero/Falso*) Permettere a più costanti di enumerazione di avere lo stesso valore può generare errori logici difficili da trovare. Nella maggior parte dei casi, le costanti di enumerazione di un `enum` dovrebbero avere valori unici in modo da evitare tali errori logici.

Risposta: Vero.

10.14 Riepilogo

Paragrafo 10.1 Introduzione

- Le strutture sono collezioni di variabili collegate sotto un unico nome. Esse possono contenere variabili di più tipi differenti di dati.
- Le strutture sono usate comunemente per definire record da memorizzare in file.
- I puntatori e le strutture possono essere usati per formare strutture di dati più complesse come liste collegate, code, pile e alberi.

Paragrafo 10.2 Definizione di strutture

- La parola chiave **struct** introduce una definizione di struttura.
- L'**etichetta della struttura** che segue la parola chiave **struct** dà il nome alla definizione di struttura e viene usata con la parola chiave **struct** per dichiarare le variabili di tipo **struct**.
- Le variabili dichiarate tra le parentesi della definizione di struttura sono i **membri** della struttura.
- I membri dello stesso tipo di struttura devono avere nomi unici.
- Ogni definizione di struttura deve terminare con un punto e virgola.
- I membri di una struttura possono avere tipi di dati primitivi o aggregati.
- Una struttura non può contenere una variabile del suo stesso tipo, ma può includere un puntatore al suo tipo.
- Una struttura contenente un membro che è un puntatore allo stesso tipo della struttura è chiamata **struttura autoreferenziale**. Le strutture autoreferenziali sono usate per costruire strutture di dati collegate.
- Le definizioni di strutture creano nuovi tipi di dati che vengono usati per definire variabili.
- Le variabili di un dato tipo di struttura possono essere dichiarate mettendo una lista di nomi di variabili separati da virgolette tra la parentesi graffa che chiude la definizione della struttura e il suo punto e virgola finale.
- Se una definizione di struttura non contiene un'etichetta, le variabili di tipo **struct** possono essere dichiarate solo nella definizione di struttura.
- Le uniche operazioni valide che possono essere eseguite su strutture sono: assegnare variabili di tipo **struct** a variabili dello stesso tipo, accedere all'indirizzo (&) di una variabile di tipo **struct**, accedere ai membri di una variabile di tipo **struct** e usare l'operatore **sizeof** per determinare la dimensione di una variabile di tipo **struct**.

Paragrafo 10.3 Inizializzazione di strutture

- Le strutture possono essere inizializzate usando **liste di inizializzatori**.
- Se vi sono meno inizializzatori nella lista rispetto ai membri della struttura, i membri restanti sono automaticamente inizializzati a 0 (o a NULL se il membro è un puntatore).
- I membri delle variabili di tipo **struct** definiti al di fuori di una definizione di funzione sono inizializzati a 0 o a NULL se non sono esplicitamente inizializzati nella definizione esterna.

Paragrafo 10.4 Accesso ai membri delle strutture con . e ->

- L'**operatore di membro di struttura** (**.**) e l'**operatore di puntatore a struttura** (**->**) sono usati per accedere ai membri di una struttura.
- L'operatore di membro di struttura accede a un membro di una struttura tramite il nome di una variabile di tipo **struct**.
- L'operatore di puntatore a struttura accede a un membro di una struttura per mezzo di un puntatore a una struttura.

Paragrafo 10.5 Uso delle strutture con le funzioni

- I membri individuali di una struttura, un'intera struttura o i puntatori a una struttura possono essere passati alle funzioni.
- Interi oggetti **struct** sono **passati per valore per impostazione predefinita**.
- Per passare una struttura per riferimento, passate il suo indirizzo. Gli array di strutture, come tutti gli altri array, vengono passati automaticamente per riferimento.
- Per **passare un array per valore**, create una **struct** con l'array come membro. Le strutture vengono passate per valore, per cui l'array viene passato per valore.

Paragrafo 10.6 `typedef`

- La parola chiave `typedef` crea sinonimi per tipi definiti in precedenza.
- I nomi per i tipi di struttura sono definiti spesso con `typedef` per creare nomi di tipo più corti.

Paragrafo 10.8 Unioni

- Un'unione è dichiarata con la parola chiave `union`. I suoi membri condividono lo stesso spazio di memoria.
- I membri di un'unione possono essere di un tipo qualsiasi di dati. L'operatore `sizeof` restituirà sempre un valore di una grandezza almeno equivalente alla dimensione in byte del membro più grande dell'unione.
- Si può fare riferimento solo a un membro di un'unione alla volta. È vostra responsabilità accedere al membro memorizzato correntemente.
- Le operazioni che possono essere eseguite su un'unione sono: assegnare un'unione a un'altra dello stesso tipo, accedere all'indirizzo (&) di una variabile di tipo `union` e accedere ai membri di un'unione usando l'operatore di membro di struttura e l'operatore di puntatore a struttura.
- Un'unione può essere inizializzata in una dichiarazione con un valore dello stesso tipo del suo primo membro.

Paragrafo 10.9 Operatori bit a bit

- I computer rappresentano tutti i dati internamente come sequenze di bit con i valori 0 o 1.
- Sulla maggior parte dei sistemi, una sequenza di 8 bit forma un byte – l'unità standard di memoria per una variabile di tipo `char`. Altri tipi di dati sono memorizzati in un numero maggiore di byte.
- Gli operatori bit a bit sono usati per manipolare i bit di operandi interi (`char`, `short`, `int` e `long`; sia con segno che senza segno). Normalmente si usano gli interi senza segno.
- Gli operatori bit a bit sono **AND (&)** bit a bit, **OR (|)** inclusivo bit a bit, **OR (^)** esclusivo bit a bit, **spostamento a sinistra (<<)**, **spostamento a destra (>>)** e **complemento (~)**.
- Gli operatori bit a bit AND, OR inclusivo e OR esclusivo confrontano i loro due operandi bit per bit. L'operatore AND bit a bit impone ogni bit nel risultato a 1 se il bit corrispondente in entrambi gli operandi è 1. L'operatore OR inclusivo bit a bit impone ogni bit nel risultato a 1 se il bit corrispondente nell'uno o nell'altro operando o in entrambi è 1. L'operatore OR esclusivo bit a bit impone ogni bit nel risultato a 1 se i bit corrispondenti di entrambi gli operandi sono differenti.
- L'operatore di spostamento a sinistra sposta a sinistra i bit del suo operando sinistro del numero di bit specificato nel suo operando destro. I bit vuoti a destra sono sostituiti con zeri; i bit spostati a sinistra vanno perduti.
- L'operatore di spostamento a destra sposta a destra i bit nel suo operando sinistro del numero di bit specificati nel suo operando destro. L'esecuzione di uno spostamento a destra su un `unsigned int` fa sì che i bit vuoti alla sinistra siano sostituiti da zeri; i bit spostati a destra vanno perduti.
- L'operatore complemento bit a bit impone a 1 tutti i bit che sono a 0 nel suo operando e a 0 tutti i bit che sono a 1 nel risultato.
- L'operatore AND bit a bit è usato spesso con un operando chiamato **maschera**, un valore intero con bit specifici impostati a 1. Le maschere sono usate per nascondere alcuni bit in un valore mentre vengono selezionati altri bit.
- `CHAR_BIT` (definita in `<limits.h>`) rappresenta il numero di bit in un byte (normalmente 8). Si può usare per rendere un programma di manipolazione di bit più generico e portabile.
- Ogni operatore bit a bit binario ha un **operatore di assegnazione bit a bit** corrispondente.

Paragrafo 10.10 Campi di bit

- Un **campo di bit** specifica il numero di bit in cui memorizzare un membro intero `unsigned` o `signed` di una struttura o di un'unione.
- Un campo di bit viene dichiarato facendo seguire il nome di un membro `unsigned int` o `int` da due punti (`:`) e da una costante intera che rappresenta la larghezza del campo. La costante deve essere un intero tra 0 e il numero totale di bit usati per memorizzare un `int` sul sistema (incluso).
- Ai membri campo di bit di strutture si accede esattamente come a qualsiasi altro membro di strutture.
- È possibile specificare un **campo di bit anonimo** da usare per **occupare lo spazio inutilizzato** in una struttura.
- Un **campo di bit anonimo con una larghezza pari a zero** allinea il successivo campo di bit su un nuovo confine di unità di memoria.

Paragrafo 10.11 Costanti di enumerazione

- Un `enum` definisce un insieme di costanti intere rappresentate da identificatori. I valori in un `enum` partono da 0, a meno che non sia specificato diversamente, e sono incrementati di 1.
- Gli identificatori in un `enum` devono essere unici.
- Il valore di una costante `enum` può essere impostato esplicitamente tramite assegnazione in una definizione `enum`.

Esercizi di autovalutazione

10.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- a) Una _____ è un insieme di variabili collegate sotto un unico nome.
- b) Un' _____ è un insieme di variabili sotto un unico nome in cui le variabili condividono la stessa memoria.
- c) I bit nel risultato di un'espressione che usa l'operatore _____ sono impostati a 1 se i bit corrispondenti in ogni operando sono impostati a 1. Altrimenti, i bit sono impostati a zero.
- d) Le variabili dichiarate in una definizione di struttura sono chiamate i suoi _____.
- e) In un'espressione che usa l'operatore _____, i bit sono impostati a 1 se almeno uno dei bit corrispondenti in entrambi gli operandi è impostato a 1. Altrimenti i bit sono impostati a zero.
- f) La parola chiave _____ introduce una dichiarazione di struttura.
- g) La parola chiave _____ crea un sinonimo di un tipo di dati definito in precedenza.
- h) In un'espressione che usa l'operatore _____, i bit sono impostati a 1 se esattamente uno solo dei bit corrispondenti in entrambi gli operandi è impostato a 1. Altrimenti i bit sono impostati a zero.
- i) L'operatore AND (&) bit a bit si usa spesso per _____ i bit, vale a dire per selezionare certi bit mentre gli altri vengono azzerati.
- j) La parola chiave _____ è usata per introdurre una definizione di unione.
- k) Il nome della struttura è detto _____ della struttura.
- l) Al membro di una struttura si accede sia con l'operatore di _____ sia con l'operatore di _____.
- m) Gli operatori di _____ e di _____ sono usati per spostare i bit di un valore, rispettivamente, a sinistra o a destra.
- n) Una _____ è un insieme di interi rappresentati da identificatori.

10.2 Stabilite se ognuna delle seguenti affermazioni è *vera* o *falsa*. Se *falsa*, spiegate perché.

- a) Le strutture possono contenere variabili di un solo tipo di dati.
- b) Due unioni possono essere confrontate (usando `==`) per determinare se sono uguali.
- c) L'etichetta di una struttura è facoltativa.
- d) I membri di strutture differenti devono avere nomi unici.
- e) La parola chiave `typedef` è usata per definire nuovi tipi di dati.

- f) Le strutture vengono sempre passate alle funzioni per riferimento.
- g) Le strutture non possono essere confrontate usando gli operatori == e !=.

- 10.3 Scrivete il codice per effettuare ognuna delle seguenti operazioni.
- a) Definire una struttura chiamata part contenente la variabile unsigned int partNumber e l'array di char partName con valori che possono essere lunghi fino a 25 caratteri (incluso il carattere nullo di terminazione).
 - b) Definire Part come sinonimo per il tipo struct part.
 - c) Usare Part per dichiarare la variabile a di tipo struct part, l'array b[10] di tipo struct part e la variabile ptr di tipo puntatore a struct part.
 - d) Leggere un numero di una parte e un nome di una parte dalla tastiera e memorizzarli nei singoli membri della variabile a.
 - e) Assegnare i valori dei membri della variabile a all'elemento 3 dell'array b.
 - f) Assegnare l'indirizzo dell'array b alla variabile puntatore ptr.
 - g) Stampare i valori dei membri dell'elemento 3 dell'array b usando la variabile ptr e l'operatore di puntatore a struttura per riferirsi ai membri.
- 10.4 Trovate l'errore in ognuno dei seguenti punti.
- a) Supponete che struct card contenga due puntatori di tipo const char * chiamati face e suit. Supponete inoltre che la variabile c sia di tipo struct card e la variabile cPtr sia un puntatore a struct card. Alla variabile cPtr è stato assegnato l'indirizzo di c.

```
printf("%s\n", *cPtr->face);
```

 - b) Supponete che struct card contenga due puntatori di tipo const char * chiamati face e suite e che l'array hearts[13] sia di tipo struct card. La seguente istruzione deve stampare il membro face dell'elemento 10 dell'array.

```
printf("%s\n", hearts.face);
```

 - c) union values {
 char w;
 float x;
 double y;
 };
 union values v = {1.27};
 - d) struct person {
 char lastName[15];
 char firstName[15];
 unsigned int age;
 }
 - e) Supponete che struct person sia stata definita come nel punto (d) ma con la correzione appropriata.

```
person d;
```

 - f) Supponete che la variabile p sia di tipo struct person e che la variabile c sia di tipo struct card.

```
p = c;
```

Risposte agli esercizi di autovalutazione

- 10.1 a) struttura. b) unione. c) AND (&) bit a bit. d) membri. e) OR (|) inclusivo bit a bit. f) struct. g) typedef. h) OR (^) esclusivo bit a bit. i) mascherare. j) union. k) etichetta. l) membro di struttura, puntatore a struttura. m) spostamento a sinistra (<<), spostamento a destra (>>). n) enumerazione.
- 10.2 a) *Falso*. Una struttura può contenere variabili di molti tipi di dati.
 b) *Falso*. Le unioni non possono essere confrontate, perché potrebbero esservi byte di dati indefiniti con valori differenti nelle variabili di tipo unione che sarebbero altrimenti identiche.

- c) *Vero.*
 d) *Falso.* I membri di strutture separate possono avere gli stessi nomi, ma i membri della stessa struttura devono avere nomi unici.
 e) *Falso.* La parola chiave `typedef` è usata per definire nomi nuovi (sinonimi) per tipi di dati definiti in precedenza.
 f) *Falso.* Le strutture vengono sempre passate alle funzioni per valore.
 g) *Vero*, a causa di problemi di allineamento.
- 10.3 a) `struct part {
 unsigned int partNumber;
 char partName[25];
};`
 b) `typedef struct part Part;`
 c) `Part a, b[10], *ptr;`
 d) `scanf("%d%24s", &a.partNumber, a.partName);`
 e) `b[3] = a;`
 f) `ptr = b;`
 g) `printf("%d %s\n", (ptr + 3)->partNumber, (ptr + 3)->partName);`
- 10.4 a) Le parentesi che devono racchiudere `*cPtr` sono state omesse, con la conseguenza che l'ordine di valutazione dell'espressione è scorretto. L'espressione deve essere:
`cPtr->face`
 o
`(*cPtr).face`
- b) L'indice dell'array è stato omesso. L'espressione deve essere `hearts[10].face`.
 c) Un'unione può essere inizializzata solo con un valore che ha lo stesso tipo del suo primo membro.
 d) È necessario un punto e virgola per terminare una definizione di struttura.
 e) La parola chiave `struct` è stata omessa dalla dichiarazione della variabile. La dichiarazione deve essere:
`struct person d;`
- f) Le variabili di tipi differenti di strutture non possono essere assegnate l'una all'altra.

Esercizi

- 10.5 Fornite la definizione per ognuna delle seguenti strutture e unioni.
- a) La struttura `inventory` contenente l'array di caratteri `partName[30]`, l'intero `partNumber`, il membro `price` in virgola mobile, l'intero `stock` e l'intero `reorder`.
 b) L'unione `data` contenente `char c, short s, long b, float f e double d`.
 c) Una struttura chiamata `address` contenente gli array di caratteri `streetAddress[25], city[20], state[3] e zipCode[6]`.
 d) La struttura `student` contenente gli array `firstName[15] e lastName[15]` e la variabile `homeAddress` di tipo `struct address` del punto (c).
 e) La struttura `test` contenente 16 campi di bit di 1 bit ciascuno. I nomi dei campi di bit sono le lettere da `a a p`.

- 10.6 Date le seguenti definizioni di strutture e di variabili:

```
struct customer {  
    char lastName[15];  
    char firstName[15];  
    unsigned int customerNumber;  
  
    struct {  
        char phoneNumber[11];
```

```

    char address[50];
    char city[15];
    char state[3];
    char zipCode[6];
} personal;
} customerRecord, *customerPtr;
customerPtr = &customerRecord;

```

scrivete un'espressione utilizzabile per accedere ai membri delle strutture in ognuna delle seguenti parti.

- a) Membro lastName della struttura customerRecord.
- b) Membro lastName della struttura puntata da customerPtr.
- c) Membro firstName della struttura customerRecord.
- d) Membro firstName della struttura puntata da customerPtr.
- e) Membro customerNumber della struttura customerRecord.
- f) Membro customerNumber della struttura puntata da customerPtr.
- g) Membro phoneNumber del membro personal della struttura customerRecord.
- h) Membro phoneNumber del membro personal della struttura puntata da customerPtr.
- i) Membro address del membro personal della struttura customerRecord.
- j) Membro address del membro personal della struttura puntata da customerPtr.
- k) Membro city del membro personal della struttura customerRecord.
- l) Membro city del membro personal della struttura puntata da customerPtr.
- m) Membro state del membro personal della struttura customerRecord.
- n) Membro state del membro personal della struttura puntata da customerPtr.
- o) Membro zipCode del membro personal della struttura customerRecord.
- p) Membro zipCode del membro personal della struttura puntata da customerPtr.

10.7 (Modifiche al programma per il mescolamento e la distribuzione di carte) Modificate il programma della Figura 10.7 per mescolare le carte usando un algoritmo di mescolamento ad alte prestazioni (come mostrato nella Figura 10.2). Stampate il mazzo risultante in un formato a due colonne che utilizzi i nomi di face e suit. Fate precedere ogni carta dal suo colore.

10.8 (Uso di unioni) Create un'unione integer con i membri char c, short s, int i e long b. Scrivete un programma che riceva in ingresso valori di tipo char, short, int e long e li memorizzi in variabili di tipo union integer. Ogni variabile di tipo union deve essere stampata come un char, un short, un int e un long. I valori sono sempre stampati correttamente?

10.9 (Uso di unioni) Create l'unione floatingPoint con i membri float f, double d e long double x. Scrivete un programma che riceva in ingresso valori di tipo float, double e long double e li memorizzi in variabili di tipo union floatingPoint. Ogni variabile di tipo union deve essere stampata come un float, un double e un long double. I valori sono sempre stampati correttamente?

10.10 (Spostamento a destra di interi) Scrivete un programma che sposti a destra una variabile intera di 4 bit. Il programma deve stampare l'intero in bit prima e dopo l'operazione di spostamento. Il vostro sistema mette degli 0 o degli 1 nei bit vuoti?

10.11 (Spostamento a sinistra di interi) Spostare a sinistra un unsigned int di 1 bit equivale a moltiplicare il valore per 2. Scrivete la funzione power2 che prende due argomenti interi number e pow e calcola

$$\text{number} * 2^{\text{pow}}$$

Usate l'operatore di spostamento per calcolare il risultato. Stampate i valori come interi e come bit.

10.12 (Impacchettare caratteri in un intero) L'operatore di spostamento a sinistra può essere usato per impacchettare quattro valori di tipo carattere in una variabile unsigned int di quattro byte. Scrivete un programma che riceva in ingresso quattro caratteri dalla tastiera e li passi alla funzione packCharacters. Per impacchettare quattro caratteri in una variabile unsigned int, assegnate il primo carattere alla variabile unsigned int, effettuate uno spostamento della variabile unsigned int a sinistra di 8 posizioni di bit e combinate la variabile unsigned con il secondo carattere usando l'operatore OR inclusivo bit a bit. Ripetete

questo processo per il terzo e il quarto carattere. Il programma deve stampare i caratteri nel loro formato in bit prima e dopo che siano impacchettati nello `unsigned int` per provare che i caratteri sono stati di fatto impacchettati correttamente.

10.13 (Spacchettare caratteri da un intero) Usando l'operatore di spostamento a destra, l'operatore AND bit a bit e una maschera, scrivete la funzione `unpackCharacters` che prenda il valore `unsigned int` dell'Esercizio 10.12 e lo spacchetti in quattro caratteri. Per spacchettare caratteri da un `unsigned int` di quattro byte, combinate il valore `unsigned int` con la maschera `4278190080` (`11111111 00000000 00000000 00000000`) e spostate completamente a destra gli 8 bit risultanti. Assegnate il valore risultante a una variabile `char`, quindi combinate il valore `unsigned int` con la maschera `16711680` (`00000000 11111111 00000000 00000000`). Assegnate il risultato a un'altra variabile `char`. Continuate questo processo con le maschere `65280` e `255`. Il programma deve stampare l'`unsigned int` in bit prima di essere spacchettato, poi stampare i caratteri in bit per confermare che sono stati spacchettati correttamente.

10.14 (Inversione dell'ordine dei bit di un intero) Scrivete un programma che inverta l'ordine dei bit in un valore `unsigned int`. Il programma deve ricevere in ingresso il valore dall'utente e chiamare la funzione `reverseBits` per stampare i bit in ordine inverso. Stampate il valore in bit sia prima che dopo l'inversione dei bit per confermare che essi siano stati invertiti correttamente.

10.15 (Funzione portabile `displayBits`) Modificate la funzione `displayBits` della Figura 10.4 in modo che essa sia portabile tra i sistemi che usano interi di 2 byte e i sistemi che usano interi di 4 byte. [Suggerimento: usate l'operatore `sizeof` per determinare la dimensione di un intero su una particolare macchina.]

10.16 (Qual è il valore di X?) Il programma seguente usa la funzione `multiple` per determinare se l'intero inserito dalla tastiera è un multiplo di qualche intero X. Analizzate la funzione `multiple`, quindi determinate il valore di X.

```

1 // ex10_16.c
2 // Questo programma determina se un valore e' un multiplo di X.
3 #include <stdio.h>
4
5 int multiple(int num); // prototipo
6
7 int main(void) {
8     int y; // y memorizza un intero inserito dall'utente
9
10    puts("Enter an integer between 1 and 32000: ");
11    scanf("%d", &y);
12
13    // se y e' un multiplo di X
14    if (multiple(y)) {
15        printf("%d is a multiple of X\n", y);
16    }
17    else {
18        printf("%d is not a multiple of X\n", y);
19    }
20}
21
22 // determina se num e' un multiplo di X
23 int multiple(int num) {
24     int mask = 1; // inizializza mask
25     int mult = 1; // inizializza mult
26
27     for (int i = 1; i <= 10; ++i, mask <= 1) {
28         if ((num & mask) != 0) {

```

```

29         mult = 0;
30         break;
31     }
32 }
33
34 return mult;
35 }
```

10.17 Cosa fa il seguente programma?

```

1 // ex10_17.c
2 #include <stdio.h>
3
4 int mystery(unsigned int bits); // prototipo
5
6 int main(void) {
7     unsigned int x; // x memorizza un intero inserito dall'utente
8
9     puts("Enter an integer: ");
10    scanf("%u", &x);
11
12    printf("The result is %d\n", mystery(x));
13 }
14
15 // Cosa fa questa funzione?
16 int mystery(unsigned int bits) {
17     unsigned int mask = 1 << 31; // inizializza mask
18     unsigned int total = 0; // inizializza total
19
20     for (unsigned int i = 1; i <= 32; ++i, bits <<= 1) {
21         if ((bits & mask) == mask) {
22             ++total;
23         }
24     }
25
26     return !(total % 2) ? 1 : 0;
27 }
```

10.18 (Algoritmo di mescolamento Fisher-Yates) Fate ricerche online sull'algoritmo di mescolamento Fisher-Yates, poi utilizzatelo per reimplementare la funzione `shuffle` della Figura 10.2.

Paragrafo speciale: casi pratici sulla programmazione di giochi con raylib

State per cominciare un viaggio emozionante e stimolante attraverso i mondi della grafica e dell'animazione, della multimedialità e dello sviluppo di giochi con la **libreria per la programmazione di giochi raylib**^{2,3}, gratuita, open source e multipiattaforma. La libreria supporta Windows, macOS, Linux e molte altre piattaforme, incluse Android, Raspberry Pi e il Web. Raylib è una libreria del C, ma può essere usata con C++, C#, Java, JavaScript, Python e molti altri linguaggi di programmazione.⁴

2. Raylib è soggetto al Copyright ©2013-2020 Ramon Santamaria (@raysan5).

3. "raylib." Accesso 14 novembre 2020. <https://www.raylib.com>.

4. "raylib bindings." Accesso 14 dicembre 2020. <https://github.com/raysan5/raylib/blob/master/BINDINGS.md>.

Nei primi tre casi pratici di questo paragrafo speciale studierete due giochi e una simulazione che abbiamo realizzato per aiutarvi ad apprendere i fondamenti di raylib.

- Nell'Esercizio 10.19, studierete il codice completo del gioco **SpotOn**, che mette alla prova i vostri riflessi chiedendovi di fare clic su punti in rapido movimento prima che spariscano. A ogni nuovo livello, la velocità dei punti aumenta, rendendo il gioco più impegnativo.
- Nell'Esercizio 10.20, studierete il codice completo del gioco **Cannon**, che vi sfida a distruggere nove obiettivi mobili prima dello scadere del tempo a disposizione. Il gioco è reso più difficile da un blocco in continuo movimento.
- Nell'Esercizio 10.21, userete una visualizzazione dinamica per “animare” la legge dei grandi numeri. Studierete il codice completo della simulazione del lancio di dadi che visualizza un grafico a barre animato. Quando la simulazione lancia il dado, vengono aggiornate le frequenze in un array. Poi il programma visualizza la frequenza di ogni faccia del dado, la sua percentuale rispetto al totale dei lanci e una barra che rappresenta la grandezza della frequenza. Per un dado a sei facce, i valori da 1 a 6 dovrebbero verificarsi con una frequenza simile: la probabilità per ciascun valore dovrebbe essere 1/6 o 16,67%. Se lanciamo un dado 6.000 volte, ci aspettiamo di vedere circa 1.000 volte ciascuna faccia. Come il lancio di una moneta, il risultato del lancio di un dado è casuale, quindi alcune facce potrebbero apparire più o meno di 1.000 volte. Aumentando il numero dei lanci, osserverete che le frequenze si avvicinano al 16,67% e le lunghezze delle barre del grafico diventano quasi identiche, confermando così la legge dei grandi numeri.

Questi giochi e simulazioni usano molte delle funzionalità di raylib, tra cui forme, testo, colori, suoni, animazione, rilevamento delle collisioni e gestione di eventi con input da parte dell'utente (come fare clic con il mouse e usare sequenze di tasti). In ogni esercizio vengono suggeriti miglioramenti che potete apportare al nostro codice.

Studiare il codice completo delle soluzioni

Una componente fondamentale per diventare un programmatore professionista consiste nel leggere e comprendere una grande quantità di codice scritto da altri. Vi capiterà molto spesso di visitare siti come GitHub.com per trovare codice open source da incorporare nei vostri progetti. Per questi primi tre casi pratici con raylib, forniamo il codice completo delle soluzioni nella sottocartella `raylib` nel Code Examples disponibile nella piattaforma MyLab.

Ogni file di codice sorgente include commenti approfonditi che:

- offrono una panoramica delle funzioni principali del codice;
- elencano le funzioni di raylib utilizzate;
- forniscono i dettagli necessari per farvi capire come funziona ciascun programma.

Compilate ed eseguite ciascun programma, giocateci e studiate attentamente il nostro codice. Sarà impegnativo ma ne varrà la pena. Lavorando con il fantastico pacchetto open source di raylib entrerete nel mondo della computer grafica e della programmazione di giochi. Formerete le basi per affrontare le modifiche al codice suggerite e gli altri esercizi sulla programmazione di giochi.

Codice di esempio nella libreria raylib

Il team di sviluppatori di raylib mette a disposizione numerosi **programmi demo in C** sul sito

<https://www.raylib.com/examples.html>

ed **esempi di giochi** sul sito

<https://www.raylib.com/games.html>

con codice sorgente completo. Per imparare altre funzionalità e tecniche di raylib potete studiare il codice sorgente completo fornito con raylib per ciascuno di questi esempi e giochi.

Implementazione di giochi e simulazioni con raylib

Utilizzando quanto appreso dal nostro codice negli Esercizi 10.19-10.21, potrete ora migliorare i giochi e la simulazione viste e crearene di nuove:

- Nell'Esercizio 10.22, reimplementerete la soluzione alla **gara tra la tartaruga e la lepre** dell'Esercizio 5.54, incorporando i suoni di una classica corsa di cavalli, l'immagine di una tartaruga e l'immagine di una lepre, e suonando l'ouverture del Guglielmo Tell in sottofondo durante la gara.
- Nell'Esercizio 10.23, reimplementerete la simulazione ad alte prestazioni per mescolare e distribuire carte del Paragrafo 10.7, usando raylib e belle immagini di pubblico dominio di carte per visualizzare un mazzo di carte.
- Negli Esercizi 10.26 e 10.27, proverete ad apportare miglioramenti alle nostre versioni dei giochi SpotOn e Cannon.
- Negli Esercizi 10.28-10.30, creerete visualizzazioni per il lancio di una moneta, per il lancio di due dadi a sei facce (che producono le somme da 2 a 12) e per mostrare i risultati di vincita/sconfitta del gioco di dadi da casinò Craps, sulla base della lunghezza dei giochi.

Negli esercizi successivi vengono proposti diversi altri giochi. Siate creativi: progettatevi e costruitevi nuovi!

Ambiente Windows autonomo di raylib

Raylib contiene un ambiente Windows autonomo con tutto quanto il necessario per creare giochi. Il pacchetto comprende:

- la libreria raylib per la programmazione di giochi;
- esempi di codice e di giochi di raylib;
- il compilatore `gcc` in MinGW⁵ (Minimalist GNU for Windows);
- l'editor di testo Notepad++, che è preconfigurato per consentirvi di compilare ed eseguire gli esempi di codice di raylib, gli esempi di gioco e i giochi da voi creati.

È possibile scaricare gratuitamente la versione MinGW di questo ambiente autonomo dal sito:

<https://raysan5.itch.io/raylib>

Per compilare ed eseguire gli esempi e i giochi campione di raylib in questo ambiente basta aprire il file C in Notepad++ e premere il tasto *F6*. In questo modo viene visualizzata una finestra con i comandi di compilazione ed esecuzione che verranno eseguiti facendo clic su **OK**. Per le applicazioni che non hanno argomenti della riga di comando, fate semplicemente clic su **OK** per compilare ed eseguire il vostro codice. Per le applicazioni con argomenti della riga di comando, come la nostra simulazione del lancio di dadi, modificate il comando **Execute program** per mettere gli argomenti della riga di comando alla fine della riga, poi fate clic su **OK**.

Installazione di raylib su Windows, macOS e Linux

I seguenti URL contengono le istruzioni per scaricare e installare raylib su Windows, macOS e Linux. Gli utenti di Windows che scelgono l'opzione dell'ambiente autonomo non hanno bisogno di eseguire queste istruzioni di installazione aggiuntive:

- Windows (per chi desidera usare raylib con altri compilatori Windows):
<https://github.com/raysan5/raylib/wiki/Working-on-Windows>
- macOS: <https://github.com/raysan5/raylib/wiki/Working-on-macOS>
- Linux: <https://github.com/raysan5/raylib/wiki/Working-on-GNU-Linux>

5. "MinGW (Minimalist GNU for Windows)." Accesso 16 dicembre 2020. <http://www.mingw.org/>.

Guida rapida di raylib

L'utilizzo di raylib è relativamente semplice, ma le sue funzioni non sono documentate in modo approfondito sul sito raylib.com. Per una lista completa delle funzioni di raylib, consultate la **guida rapida (cheat sheet) di raylib**:

<https://www.raylib.com/cheatsheet/cheatsheet.html>

Ogni funzione viene elencata con il suo prototipo seguito da un commento che ne spiega brevemente lo scopo. Il cheat sheet contiene inoltre i nomi dei tipi personalizzati e delle costanti di colore di raylib. I nomi delle funzioni di raylib iniziano con una lettera maiuscola, mentre per convenzione in C i nomi delle funzioni iniziano con una lettera minuscola.

Il file di intestazione raylib.h su GitHub

Quando si lavora con software open source, potrebbe capitare di dover guardare il codice sorgente per avere una risposta alle proprie domande. Per esempio, raylib definisce molti dei propri tipi (tipicamente come `struct` o `enum`), e la maggior parte essi non è elencata nel cheat sheet. Tuttavia, il codice sorgente completo di raylib è disponibile nel suo repository su GitHub:

<https://github.com/raysan5/raylib/>

Il file di intestazione `raylib.h` contiene le definizioni dei tipi di raylib:

<https://github.com/raysan5/raylib/blob/master/src/raylib.h>

Tra i tipi di raylib che userete, sono inclusi i seguenti.

- **Vector2**: contiene i membri `x` e `y` che rappresentano una coppia di coordinate `x-y`.
- **Rectangle**: contiene i membri `x`, `y`, `width` e `height` che rappresentano l'angolo superiore sinistro, la larghezza e l'altezza di un rettangolo.
- **Color**: i colori in raylib sono definiti usando i **colori RGBA (Red Blue Green Alpha)**. Ogni colore ha componenti di rosso (`r`), verde (`g`), blu (`b`) e alpha (`a`; trasparenza) con valori nell'intervallo 0-255. Nel cheat sheet di raylib si trova un elenco delle costanti dei colori predefiniti di raylib. È anche possibile specificare colori personalizzati creando oggetti `Color` e impostando i loro membri `r`, `g`, `b` e `a`.
- **Sound**: contiene membri che memorizzano i suoni caricati in memoria con la funzione di raylib `LoadSound`.
- **Texture2D**: contiene membri che rappresentano una texture caricata nella memoria del processore grafico (GPU).

Per questi casi pratici di raylib non è necessario conoscere i dettagli dei tipi `Sound` e `Texture2D`, ma se siete curiosi potete vederne le definizioni in `raylib.h`.

Animazioni "frame-by-frame" di raylib

Nei giochi di raylib, un **ciclo di gioco** realizza un'**animazione frame-by-frame** (un fotogramma alla volta). Ogni iterazione del ciclo esegue due passaggi.

1. **Aggiorna gli elementi del gioco per il frame di animazione successivo**: in questa fase, si implementa la logica di gioco che determina i nuovi stati degli elementi del gioco. Qui viene implementata la logica dell'esperienza di gioco. Le attività eseguite in questa fase includono: aggiornare le posizioni degli elementi, verificare se ci sono eventi di input da parte dell'utente (come clic fatti con il mouse), rilevare collisioni tra elementi del gioco, aggiornare il punteggio, controllare se il gioco è finito, ecc. Le posizioni degli elementi sono specificate da coppie di coordinate `x-y` con valori compresi nelle dimensioni di larghezza e altezza dello schermo (`0,0` è l'angolo in alto a sinistra).
2. **Disegna gli elementi del gioco per il frame di animazione successivo**: in questa fase, si usano le funzioni di disegno di raylib per disegnare gli elementi del gioco nelle loro posizioni correnti. Raylib memorizza i pixel del nuovo frame di animazione in una memoria temporanea (detta **buffer fuori schermo**). Quando è terminata la fase di disegno, raylib visualizza il contenuto del buffer fuori schermo, sostituendo il frame di animazione precedente sullo schermo.

Struttura dei giochi di raylib

Un tipico gioco realizzato con la libreria raylib contiene la struttura seguente nella sua funzione `main`, la cui spiegazione si trova dopo il listato.

```

1 int main(void) {
2     // inizializzazione
3     InitWindow(screenWidth, screenHeight, "Window Title");
4     InitGame();
5     SetTargetFPS(60);
6
7     // ciclo di gioco
8     while (!WindowShouldClose()) {
9         UpdateGame(); // aggiorna gli elementi del gioco
10        DrawGame(); // disegna il frame di animazione successivo
11    }
12
13    // ripulitura
14    UnloadGame(); // rilascia le risorse del gioco
15    CloseWindow(); // chiudi la finestra di gioco
16 }
```

- La funzione di raylib `InitWindow` (riga 3) specifica la larghezza e l'altezza della finestra di gioco in pixel e il suo titolo.
- Un tipico esempio di gioco di raylib contiene una funzione `InitGame` definita dall'utente (riga 4), dove si possono caricare suoni, texture e immagini e vengono inizializzati gli elementi del gioco e le variabili che mantengono lo stato del gioco. Quando un gioco termina e l'utente decide di giocare di nuovo, di norma si chiama la funzione `InitGame` per resettare lo stato del gioco prima di iniziare uno nuovo.
- La funzione di raylib `SetTargetFPS` (riga 5) specifica il numero di frame di animazione al secondo che raylib prova a disegnare (più è alto il numero di frame al secondo e più l'animazione risulta fluida). I giochi per console moderni generalmente cercano di visualizzare 60 frame al secondo (*FPS, Frame Per Second*), anche se ci sono giochi che ne usano di più o di meno. Si consiglia un minimo di 30 FPS per ottenere un'animazione fluida.
- Il ciclo di gioco principale (righe 8-11) realizza gli aggiornamenti e l'animazione del gioco. Questo ciclo itera fino a quando la funzione `WindowShouldClose` di raylib ritorna vera, ovvero quando l'utente chiude la finestra di gioco o preme il tasto *Esc*. Questo ciclo aggiorna gli elementi del gioco e poi li disegna. Molti esempi di giochi di raylib mettono il codice per l'aggiornamento in una funzione chiamata `UpdateGame` e il codice per disegnare in una funzione chiamata `DrawGame`. In questo modo la manutenzione del codice è più semplice.
- Quando il ciclo di gioco termina, la funzione `UnloadGame` (riga 14) scarica tutte le risorse di gioco che sono state caricate in `InitGame`, come suoni, texture e immagini.
- La funzione di raylib `CloseWindow` (riga 15) rilascia le risorse della finestra di gioco e la chiude. Quindi, l'applicazione termina.

In questo codice, `InitGame`, `UpdateGame`, `DrawGame` e `UnloadGame` sono funzioni definite dall'utente che determinano la logica di gioco. Gli esempi di codice e di giochi di raylib solitamente usano questi nomi, che seguono la stessa convenzione usata per le funzioni di raylib che prevede l'uso della maiuscola per la prima lettera del nome. Usiamo gli stessi nomi o nomi simili nei nostri giochi e nomi simili nelle simulazioni che non siano giochi (es. `InitSimulation` invece di `InitGame`). Definiamo tutte le altre funzioni di supporto con le nostre consuete convenzioni di denominazione delle funzioni usate nel libro.

Variabili globali e costanti

Per ragioni di prestazioni, nei giochi di raylib le variabili degli elementi del gioco e dello stato del gioco sono definite come variabili globali `static`. Tali variabili sono note soltanto dalle loro definizioni fino alla fine del file nel quale sono definite. L'utilizzo di variabili globali `static` consente alle funzioni del gioco di accedere agli elementi e allo stato del gioco direttamente senza passarli come argomenti alle funzioni. In questo modo si elimina l'overhead dei meccanismi di chiamata/ritorno delle funzioni. Come vedrete, anche i giochi relativamente semplici di solito hanno numerose variabili di elementi e di stato del gioco. Definire funzioni con un gran numero di parametri rende più difficile la manutenzione, la modifica e il debugging del codice.

Come affrontare gli esercizi

Per ciascuno dei primi tre esercizi con raylib, viene descritto cosa fa il gioco e vengono mostrate schermate del gioco in svolgimento. Per ogni gioco, dovete:

1. leggere la descrizione dell'esercizio per farvi un'idea del gioco o della simulazione;
2. compilare ed eseguire diverse volte il gioco o la simulazione (se si tratta di un gioco, provate a giocarlo per vedere come funziona);
3. esplorare i programmi con codice completo e commenti da noi forniti;
4. modificare il codice ed eseguirlo nuovamente per osservare gli effetti delle vostre modifiche.

In genere, il nostro codice inizia con commenti che offrono una panoramica delle funzioni di gioco da noi scritte, un riassunto delle funzioni di raylib utilizzate e altro ancora.

Interagire con la community di raylib

Ecco alcuni dei siti più importanti⁶ dove potrete trovare video su raylib e interagire con altri utenti di raylib:

- Discord: <https://discord.gg/VkzNHUE>
- Twitter: <http://www.twitter.com/raysan5>
- Twitch: <http://www.twitch.tv/raysan5>
- Reddit: <https://www.reddit.com/r/raylib>
- Patreon: <https://www.patreon.com/raylib>
- YouTube: <https://www.youtube.com/c/raylib>

Generatore di effetti sonori rFXGen di raylib

Raylib dispone di numerosi strumenti online per creare gli elementi dei vostri giochi, tra cui icone, texture, elementi e layout dell'interfaccia grafica utente ed effetti sonori:

<https://raylibtech.itch.io/>

Abbiamo usato il generatore di effetti sonori online rFXGen di raylib:

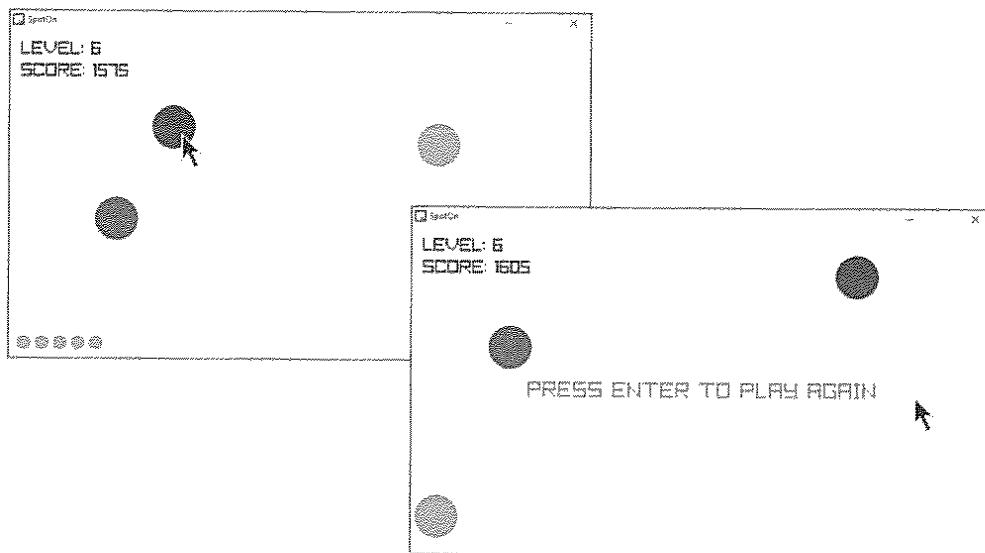
<https://raylibtech.itch.io/rfxgen>

per creare gli effetti sonori dei nostri giochi. Potete usare questi effetti sonori già forniti oppure crearne di nuovi.

Caso pratico sulla programmazione di giochi: il gioco SpotOn

10.19 (Caso pratico sulla programmazione di giochi: il gioco SpotOn) In questo esercizio relativo alla programmazione di giochi, studierete il gioco **SpotOn**, che mette alla prova i vostri riflessi chiedendovi di fare clic su punti in rapido movimento prima che spariscano.

6. "README.md." Accesso 16 dicembre 2020. <https://github.com/raysan5/raylib/README-.md>.



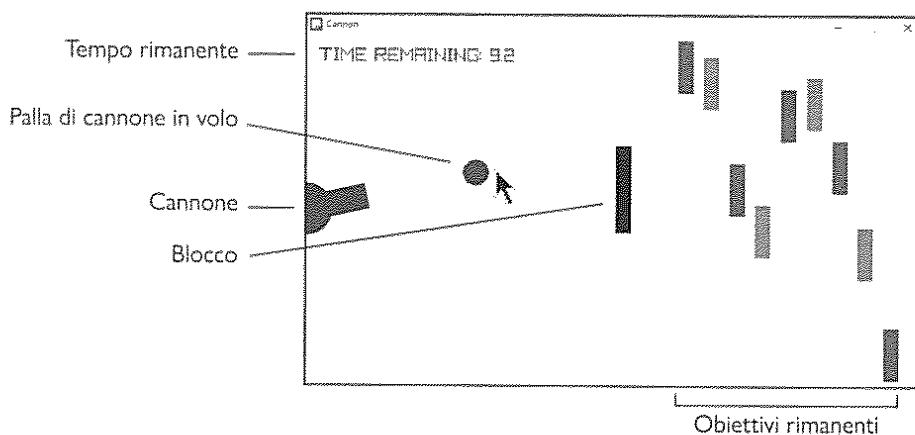
Il gioco incomincia al primo livello mostrando tre punti colorati disposti a caso. Essi si muovono con velocità e direzione casuali. Il giocatore passa di livello ogni volta che arriva a fare clic su 10 punti (e la velocità dei punti aumenta del 5%, rendendo il gioco sempre più impegnativo). Quando si fa clic su un punto, l'applicazione emette uno scoppietto e il punto sparisce. Il punteggio per ogni punto cliccato corrisponde a 10 volte il numero del livello corrente. La precisione è essenziale: ogni volta che si fa clic senza prendere un punto viene riprodotto il suono di una pernacchia e il punteggio diminuisce di 15 volte il numero del livello corrente. Il livello corrente e il punteggio sono visualizzati nell'angolo superiore sinistro dello schermo.

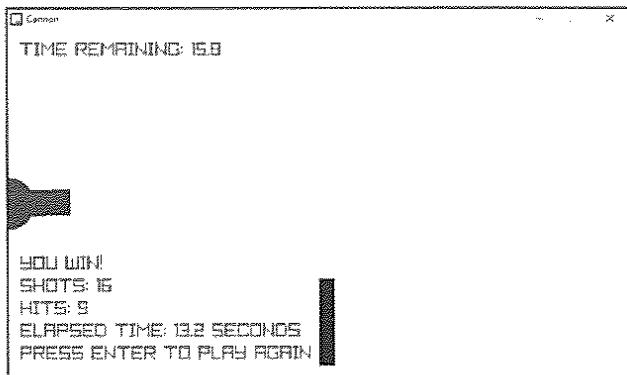
Si inizia il gioco con tre vite a disposizione, visualizzate sotto forma di tre piccoli cerchi nell'angolo inferiore sinistro dello schermo. Se un punto sparisce prima di riuscire a fare clic su di esso, si sente il suono "whoosh" e si perde una vita. Si guadagna invece una vita a ogni nuovo livello raggiunto, fino a un massimo di sette vite. Quando non si hanno più vite, il gioco finisce.

Compilate ed eseguite il gioco e provate a giocare per diverse volte. In seguito, studiate il codice di questo gioco (inclusi gli estesi commenti). Provate a modificare il codice per vedere come le modifiche apportate influiscono sul gioco. Per esempio, potete cambiare il valore della costante `spotSpeed` per far muovere i punti più o meno velocemente. Infine, migliorate il gioco implementando i suggerimenti contenuti nell'Esercizio 10.26.

Caso pratico sulla programmazione di giochi: il gioco Cannon

10.20 (Caso pratico sulla programmazione di giochi: il gioco Cannon) Nel gioco Cannon, dovete distruggere nove obiettivi prima che scadano i dieci secondi di tempo che avete a disposizione:





Il gioco ha quattro tipi di elementi visuali:

- un **cannone** controllato dal giocatore;
- una **palla di cannone**;
- **nove obiettivi** che si muovono su e giù a diverse velocità;
- un **blocco** che si muove su e giù per difendere gli obiettivi.

Gli obiettivi e il blocco si muovono in verticale a velocità differenti ma prefissate, cambiando direzione quando giungono al limite superiore o inferiore dello schermo.

Per sparare con il cannone, fate clic con il mouse. Il cannone ruota verso il punto in cui è stato fatto clic, spara un palla che si muove velocemente in linea retta in quella direzione ed emette il **suono di un'esplosione**. Sullo schermo ci può essere solo una palla di cannone alla volta.

Ogni volta che si distrugge un obiettivo, si sente il **suono di distruzione dell'obiettivo**, l'obiettivo sparisce e il tempo rimanente aumenta di tre secondi (bonus). Il blocco non può essere distrutto. Quando la palla di cannone colpisce il blocco, si sente un **suono di blocco colpito**, la palla rimbalza indietro e il tempo rimanente diminuisce di due secondi (penalità).

Si vince distruggendo tutti e nove gli obiettivi prima dello scadere del tempo. Se il timer arriva a zero, si perde. Alla fine del gioco, l'applicazione visualizza se avete vinto o perso, il numero dei colpi sparati e il tempo trascorso.

Compilate ed eseguite il gioco e provate a giocare per diverse volte. In seguito, studiate il codice di questo gioco (inclusi gli estesi commenti). In questa applicazione sono richiesti elementi di trigonometria per:

- determinare il punto finale della canna del cannone, in base alla sua angolazione;
- determinare gli incrementi x e y della palla di cannone usati per muovere la palla in ogni frame di animazione (anch'essi si basano sull'angolazione della canna del cannone).

Abbiamo pensato noi ai calcoli di trigonometria.

Provate a modificare il codice per vedere come le modifiche apportate influiscono sul gioco. Per esempio, potete modificare la velocità della palla di cannone. Infine, migliorate il gioco implementando i suggerimenti contenuti nell'Esercizio 10.27.

Visualizzazioni con raylib – Animazione della legge dei grandi numeri

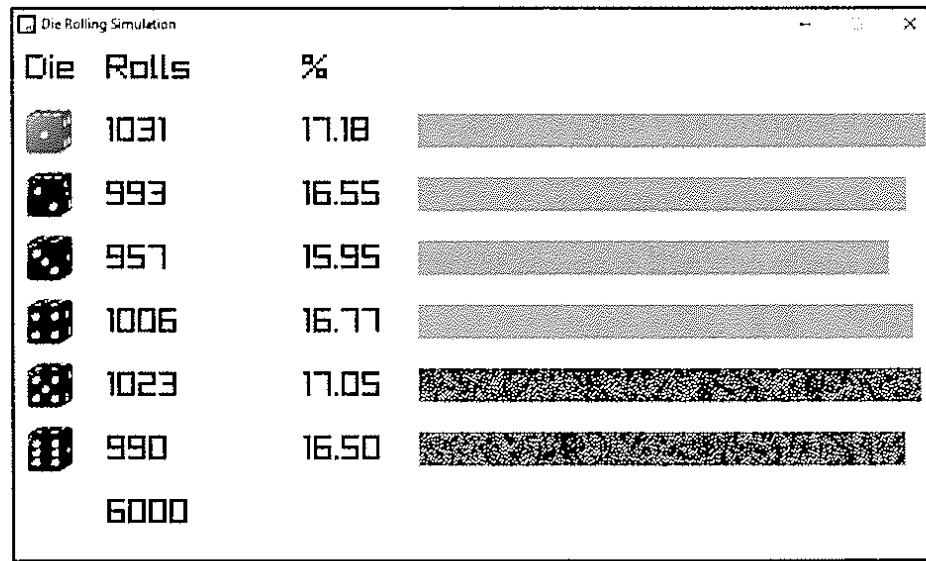
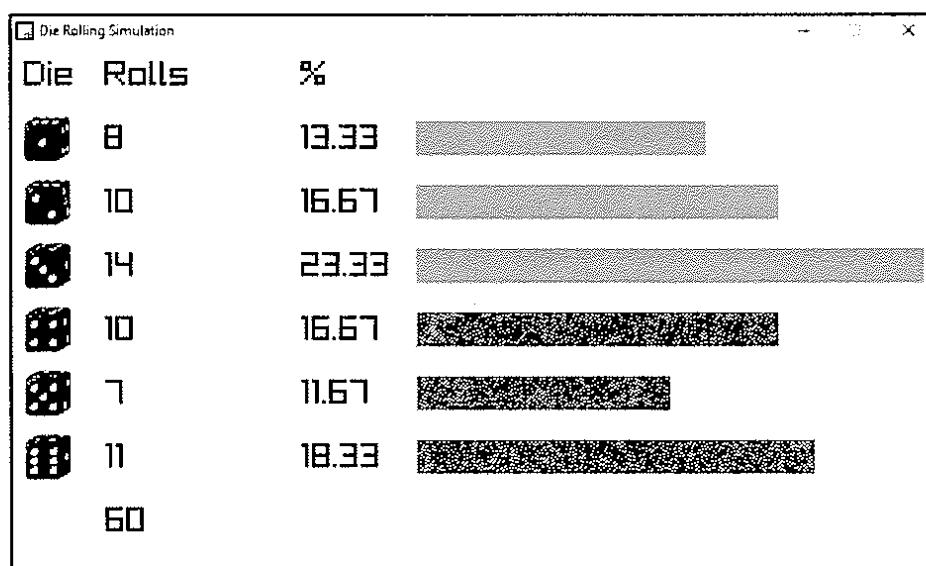
10.21 (Animazione della legge dei grandi numeri) Nei Paragrafi 5.10 e 6.4.7, abbiamo usato la generazione di numeri casuali per simulare il lancio di un dado a sei facce. In questo esercizio con raylib, userete la visualizzazione dinamica per “animare” la legge dei grandi numeri^{7,8} con una simulazione del lancio di un dado che mostrerà un grafico a barre animato. Mentre la simulazione lancia ripetutamente il dado, aggiorna un array con

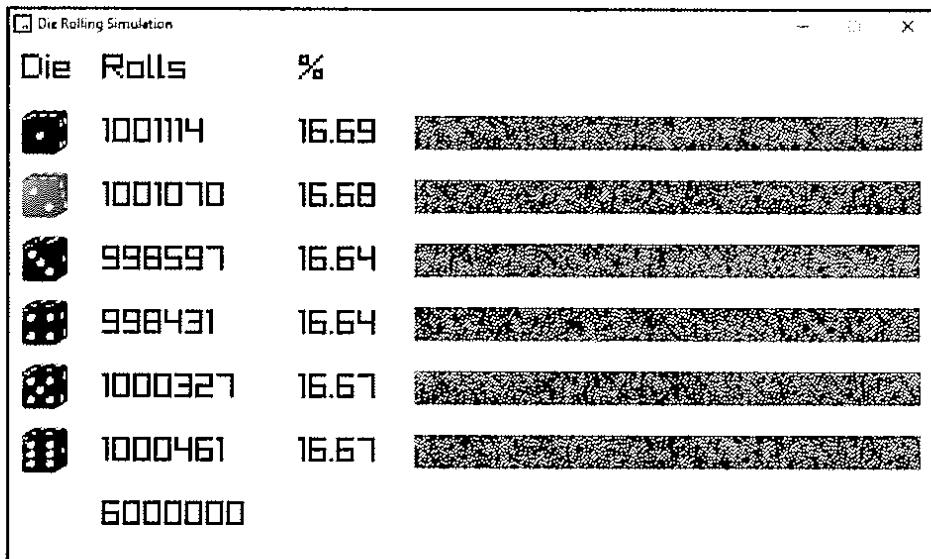
7. “Law of large numbers.” Accesso 18 dicembre 2020. https://encyclopediaofmath.org/index.php?title=Law_of_large_numbers.

8. “Law of large numbers.” Accesso 18 dicembre 2020. https://en.wikipedia.org/wiki/Law_of_large_numbers.

le frequenze di apparizione di ciascuna faccia. Poi visualizza la frequenza di ogni faccia del dado, la sua percentuale sul totale dei lanci e una barra la cui lunghezza rappresenta la dimensione del valore della frequenza.

Per un dado a sei facce, i valori da 1 a 6 dovrebbero verificarsi con una frequenza simile: la probabilità che ogni faccia ha di apparire dovrebbe essere $1/6$ o il 16,67% per ciascun lancio. Se lanciamo un dado 6.000 volte, ci aspettiamo di vedere circa 1.000 volte ciascuna faccia. Come il lancio di una moneta, il risultato del lancio di un dado è casuale, quindi alcune facce potrebbero apparire più o meno di 1.000 volte. Aumentando il numero dei lanci, secondo la legge dei grandi numeri le frequenze dovrebbero avvicinarsi al valore atteso di 16,67%. Quindi le lunghezze delle barre del grafico dovrebbero diventare quasi identiche, come mostrato nelle schermate seguenti che rappresentano il lancio di un dado per 60, 6.000 e 6.000.000 di volte.





Eseguire la simulazione su macOS o Linux

Quando si esegue questa simulazione, sono necessari due argomenti della riga di comando che rappresentano:

- la lunghezza della simulazione espressa in frame di animazione;
- il numero di lanci di dado per frame di animazione.

Se chiamiamo `RollDieDynamic` il file eseguibile del programma, il comando seguente di macOS o Linux eseguirà la simulazione per 60 frame di animazione, effettuando un lancio di dado a ogni frame, per un totale di 60 lanci:

```
./RollDieDynamic 60 1
```

In modo simile, il comando seguente eseguirà la simulazione per 600 frame di animazione, effettuando 1.000 lanci di dado a ogni frame, per un totale di 600.000 lanci:

```
./RollDieDynamic 600 1000
```

Sebbene tratteremo i dettagli degli argomenti della riga di comando solo nel Paragrafo 15.3, in questa simulazione completamente codificata sono incluse tutte le istruzioni necessarie per ricevere gli argomenti della riga di comando.

Eseguire la simulazione in ambiente Windows autonomo

Nell'ambiente Windows autonomo di raylib, per eseguire questa simulazione dovete osservare la seguente procedura.

1. Aprite `RollDieDynamic.c` in Notepad++.
2. Premete il tasto *F6*.
3. Nella finestra di dialogo **Execute**, modificate l'ultima riga dei comandi di compilazione ed esecuzione in modo che includano gli argomenti della vostra riga di comando, come in:

```
cmd /c IF EXIST ${NAME_PART}.exe ${NAME_PART}.exe 600 1000
```

Notepad++ sostituisce `$(NAME_PART)` con il nome del file in esecuzione (in questo caso `RollDieDynamic`).

4. Fate clic su **OK** per compilare ed eseguire il programma.

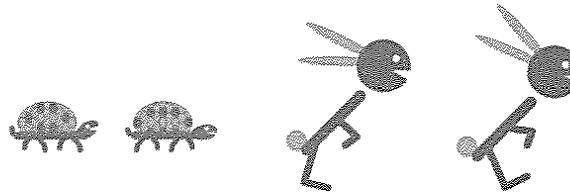
Eseguire il programma più volte

Compilate la simulazione ed eseguitela più volte, variando gli argomenti della riga di comando. Successivamente, studiate il codice della simulazione (inclusi gli estesi commenti). Come nei nostri giochi di raylib, in qualunque momento potete mettere in pausa la simulazione premendo il tasto *P* e riassumerla premendolo nuovamente. Una volta che avrete studiato il codice, provate a fare gli Esercizi 10.28-10.30, nei quali creerete le visualizzazioni per il lancio di una moneta, per il lancio di due dadi a sei facce (che producono le somme da 2 a 12) e per mostrare i risultati vincita/perdita del gioco Craps, sulla base della lunghezza dei giochi. Potete provare a utilizzare le tecniche apprese per analizzare i risultati di popolari giochi di carte come blackjack e le diverse versioni di poker.

Caso pratico: la tartaruga e la lepre con raylib – una “rappresentazione spettacolare” multimediale

10.22 (Rappresentazione multimediale della gara tra la tartaruga e la lepre con raylib) In questo esercizio utilizzerete le funzionalità di grafica, animazione e suono di raylib degli Esercizi 10.19-10.21 per arricchire la gara tra la tartaruga e la lepre dell’Esercizio 5.54. Vi aggiungerete i suoni di una tipica corsa di cavalli, nonché numerose immagini della lepre e della tartaruga, per creare una divertente “rappresentazione spettacolare” multimediale animata. In Code Examples nella piattaforma MyLab c’è una sottocartella **resources** contenente le seguenti clip audio e immagini da usare nella vostra implementazione:⁹

- una nostra registrazione audio dell’assolo di tromba “Call to Post” che viene suonato all’inizio di una corsa di cavalli;
- il suono dello sparo di un cannone che abbiamo creato per il nostro gioco raylib **Cannon**; potete usare il generatore di suoni rFXGen di raylib (<https://raylibtech.itch.io/rfxgen>) per creare voi stessi il suono di uno sparo;
- una nostra audio clip di un annunciatore che dice “And they’re off!”; potete registrare la vostra voce e dire voi questa e altre frasi da far sentire durante la gara, come “Tortoise pulls ahead!”, “Hare pulls ahead!”, “Down the stretch they come!” ecc.;
- una registrazione audio Wikimedia di pubblico dominio dell’ouverture del Guglielmo Tell¹⁰, che abbiamo ridotto alla sola parte del galoppo (bada bum, bada bum, bada bum bum bum...) e inserito nella cartella **resources** di questo capitolo in modo che possiate trasmetterla durante la gara;
- due immagini della tartaruga e due immagini della lepre, da noi elaborate, leggermente differenti tra loro;



Passando velocemente da un’immagine all’altra abbiamo creato un semplice effetto animato della corsa di questi animali. Potete vedere le animazioni nelle immagini GIF animate (`tortoise.gif` e `hare.gif`) fornite nella cartella **resources** con gli esempi di questo capitolo. Usate pure queste immagini oppure divertitevi a crearne la vostra versione.

Implementazione della gara

Implementate la gara usando la struttura di base dei giochi raylib appresa negli esercizi precedenti, effettuando i compiti seguenti.

9. Abbiamo creato questi audio e queste immagini, ma se preferite potete cercarne altri nel Web oppure crearne di nuovi. Assicuratevi di rispettare i termini di licenza per qualsiasi media utilizzerete nelle vostre applicazioni.

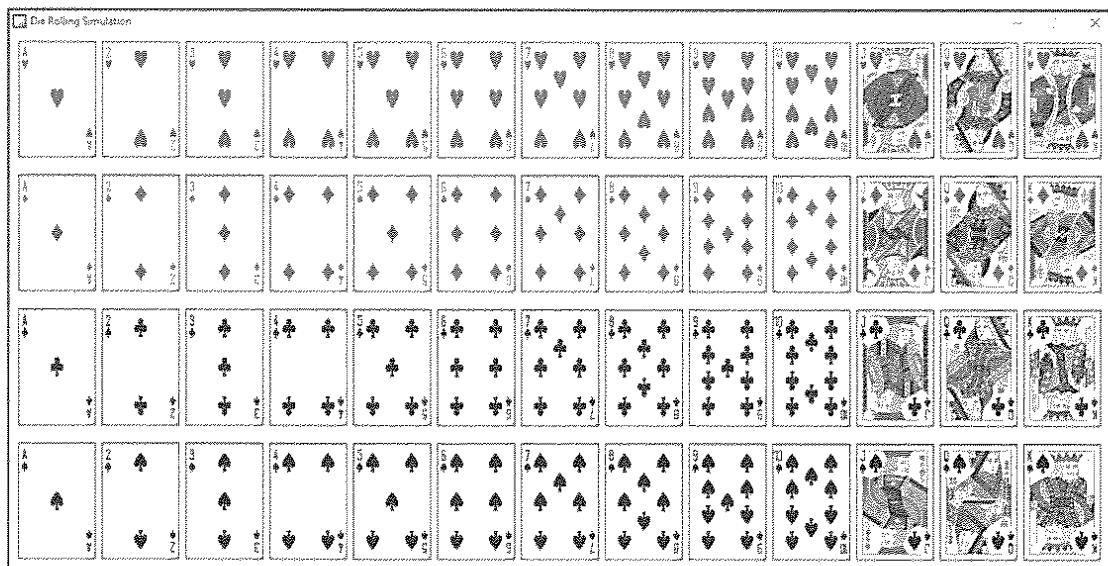
10. “File:Gioachino Rossini, William Tell Overture (military band version, 2000).ogg.” Accesso 2 gennaio 2021. [https://commons.wikimedia.org/wiki/File:Gioachino_Rossini,_William_Tell_Overture_\(military_band_version,_2000\).ogg](https://commons.wikimedia.org/wiki/File:Gioachino_Rossini,_William_Tell_Overture_(military_band_version,_2000).ogg).

- Prima dell'inizio della gara, riprodotte l'audio dell'assolo di tromba "Call to Post", che segnala a tutti i concorrenti di posizionarsi ai blocchi di partenza; mentre suona il "Call to Post", la tartaruga e la lepre devono comparire dalla parte sinistra dello schermo e prendere posizione.
- Riprodotte lo sparo del cannone per far iniziare la gara, seguito dall'annunciatore che dice "And they're off!"; a questo punto inizia l'animazione.
- Durante tutta la gara, fate suonare ripetutamente in sottofondo la parte del galoppo dell'ouverture del Guglielmo Tell (per imparare come riprodurre la musica in sottofondo consultate il codice di esempio di raylib all'indirizzo https://www.raylib.com/examples-/web/audio/loader.html?name=audio_music_stream).
- Mentre la tartaruga e la lepre si muovono sullo schermo, alternate le due immagini di ciascun animale per rendere l'effetto della corsa. La tartaruga si muove più piano della lepre, perciò passate da un'immagine all'altra della tartaruga più lentamente rispetto a quanto fate con le immagini della lepre. Quando la lepre dorme, fermate l'alternanza delle sue immagini.
- Quando la tartaruga e la lepre sono sulla stessa posizione, visualizzate "OUCH!" per indicare che la tartaruga morde la lepre, ed eventualmente riprodotte anche una clip audio con un sonoro "OUCH!".
- Se vince la tartaruga, visualizzate "Tortoise wins!" e, se volete, riprodotte una clip audio "Tortoise wins!" seguita da un'ovazione; se vince la lepre, visualizzate "Hare wins" e, se volete, riprodotte una clip audio "Hare wins" seguita da fischi. Se la gara finisce con un pareggio, potete favorire la tartaruga, in quanto svantaggiata, oppure far annunciare "It's a tie!".
- Potete riprodurre suoni di applausi e fischi e altri commenti dell'annunciatore in modo appropriato secondo l'andamento della gara; è possibile trovare online rumori di folla di pubblico dominio.

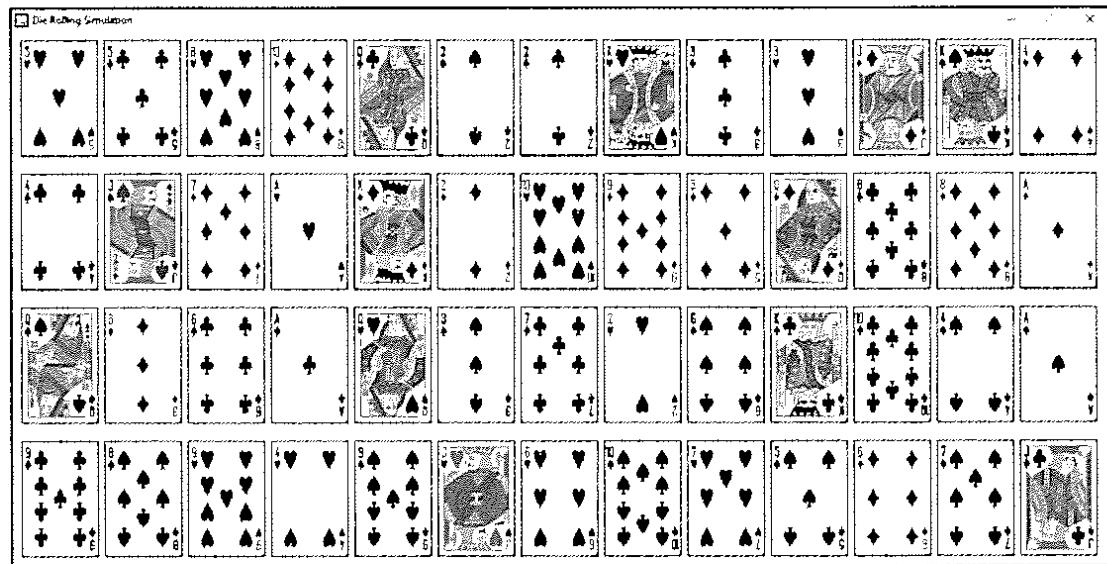
Caso pratico sulla simulazione di numeri casuali: mescolamento e distribuzione di carte ad alte prestazioni con immagini di carte e raylib

10.23 (Mescolamento e distribuzione di carte con immagini di carte e raylib) Nel Paragrafo 10.7, avete implementato una simulazione ad alte prestazioni del mescolamento e della distribuzione di carte utilizzando un array di oggetti Card. In questo esercizio, integrerete le funzionalità di raylib nella vostra simulazione, usandole per visualizzare belle immagini di carte di pubblico dominio per ciascuna carta del mazzo. Dopo aver completato questo esercizio, sarete in possesso delle competenze fondamentali per iniziare a implementare i vostri giochi di carte preferiti e per ottimizzare le soluzioni elaborate negli esercizi sui giochi di carte dei capitoli precedenti.

Caricherete le 52 immagini di carte non mescolate come oggetti Texture2D di raylib, poi le visualizzerete in una griglia 4-per-13:



Ogni volta che l'utente fa clic con il mouse, mescolate le carte e visualizzatele nuovamente:



Immagini di carte di pubblico dominio da Wikimedia Commons

Abbiamo scaricato queste immagini di carte di pubblico dominio¹¹ da:

https://commons.wikimedia.org/wiki/Category:SVG_English_pattern_playing_cards

mettendole a disposizione nella sottocartella `card_images` nel Code Examples di questo capitolo disponibile nella piattaforma MyLab. Abbiamo denominato ciascun file delle immagini usando il valore (*face*) e il seme (*suit*) della carta. Per esempio, le immagini delle picche (Spades) hanno i seguenti nomi:

- `Ace_of_Spades.png`
- `Deuce_of_Spades.png`
- `3_of_Spades.png`
- ...
- `Jack_of_Spades.png`
- `Queen_of_Spades.png`
- `King_of_Spades.png`

Implementazione della simulazione

Usate il framework di gioco basilare di raylib appreso negli Esercizi 10.19-10.21 e le tecniche di elaborazione delle immagini apprese nell'Esercizio 10.21 per portare a termine i compiti seguenti.

- a) Modificate la definizione di `struct card` della Figura 10.2 in modo da includere un membro `Texture2D` chiamato `image`, che memorizzerà le informazioni di raylib sull'immagine della carta caricata.
- b) Quando comincia l'esecuzione dell'applicazione, inizializzate il mazzo di carte non mescolato nella funzione `InitSimulation` della vostra applicazione raylib. Modificate il codice che inizializza il vostro array `deck` per caricare l'immagine di ogni carta. Potete usare le funzionalità di elaborazione delle stringhe per unire le stringhe del valore (*face*) e del seme (*suit*) di ciascuna carta e formare il nome del file che contiene l'immagine della carta usando il formato:

`face_of_suit.png` (sostituendo *face* e *suit* con il rispettivo valore e seme);

11. <https://creativecommons.org/publicdomain/zero/1.0/deed.en>.

- c) La prima volta che viene eseguita la funzione `DrawFrame` dell'applicazione `raylib`, visualizzate l'array non mescolato dei 52 oggetti `Card`, come mostrato in precedenza; dovete eseguire i calcoli necessari per determinare le coordinate `x-y` relative all'angolo superiore sinistro di ciascuna immagine.
- d) Nella funzione `Update` dell'applicazione `raylib`, verificate se l'utente ha fatto clic con il tasto sinistro del mouse e, in tal caso, mescolate le carte; la chiamata successiva alla funzione `DrawFrame` mostrerà il mazzo mescolato.

Suggerimenti per il disegno

I suggerimenti seguenti vi aiuteranno a implementare la vostra simulazione.

- In questo esercizio, per fornire ulteriore spazio e visualizzare meglio le immagini delle carte, impostate le dimensioni della finestra di `raylib` con `screenWidth` a 1280 e `screenHeight` a 620.
- Quando disegnate le immagini, impostate a 0.25 l'argomento `scale` della funzione `DrawTextureEx` di `raylib`; questo ridurrà le dimensioni delle singole immagini, e avrete spazio a sufficienza per disegnare 4 righe di 13 carte ciascuna senza che le immagini si sovrappongano.
- Dopo aver disegnato ciascuna immagine, usate la funzione `DrawRectangleLines` di `raylib` come mostrato qui sotto per inserire così un bordo nero intorno all'immagine in contrasto con lo sfondo bianco della finestra:

```
DrawRectangleLines(x, y, deck[i].image.width * scale,
                   deck[i].image.height * scale, BLACK);
```

Impostate la variabile `scale` a 0.25 per disegnare il rettangolo con la stessa scala dell'immagine.

Esercizi supplementari con raylib

10.24 (*Demo di raylib*) Compilate, eseguite e interagite con i numerosi esempi inclusi in `raylib`, disponibili nella sottocartella `examples` della cartella `raylib`. Studiate il codice sorgente fornito per ciascun esempio per approfondire la conoscenza delle funzionalità di `raylib`.

10.25 (*Esempi di giochi di raylib*) Compilate, eseguite e interagite con i numerosi esempi di giochi contenuti nella sottocartella `games` della cartella `raylib`. Studiate il codice sorgente fornito per ciascun esempio per approfondire la conoscenza delle funzionalità di `raylib`. Siate creativi: provate a modificare i giochi apportando i vostri miglioramenti.

10.26 (*Progetto: gioco SpotOn migliorato*) Provate ad apportare alcune delle seguenti modifiche al gioco `SpotOn`.

- a) Mostrate un numero maggiore di punti nei livelli più alti.
- b) Usate incrementi di velocità più alti, possibilmente casuali.
- c) Concedete un bonus per la distruzione di più punti con un solo clic.
- d) Usate punteggi diversi a seconda del colore dei punti.
- e) Rendete i punti più inafferrabili, facendo in modo che possano lampeggiare, cambiare spontaneamente direzione e dimensione, muoversi lungo percorsi non lineari.
- f) Intervallate punti più difficili da selezionare con altri più facili.
- g) Quando l'utente fa clic su un punto, animatene la fase di distruzione; per esempio, il punto potrebbe trasformarsi in cerchi concentrici che svaniscono partendo da quelli più esterni, oppure potrebbe trasformarsi in quattro fette di pizza che si allontanano dal centro del punto fino a scomparire.
- h) Riproducete il suono di una sirena quando si passa al livello successivo del gioco.
- i) Fate apparire testi che sottolineino eventi significativi come la perdita o il guadagno di una vita; il testo potrebbe rimanere visibile sullo schermo per un breve periodo e poi svanire.
- j) Aggiungete un punto di un colore speciale, che si muove a forte velocità; fare clic su questo punto distrugge tutti gli altri punti sullo schermo e offre al giocatore un bonus consistente.

10.27 (Progetto: gioco Cannon migliorato) Provate a eseguire una buona parte delle seguenti modifiche del gioco **Cannon**.

- Visualizzate una linea tratteggiata che mostra la traiettoria della palla di cannone.
- Riproducete un suono quando il blocco colpisce il bordo superiore o inferiore dello schermo.
- Riproducete un suono quando un obiettivo colpisce il bordo superiore o inferiore dello schermo.
- Aggiungete nel gioco i livelli, e in ciascun livello aumentate il numero degli obiettivi.
- Segnate il punteggio; aumentate il punteggio dell'utente per ogni obiettivo colpito moltiplicando per 10 il numero del livello corrente; diminuite il punteggio di 15 volte il numero del livello corrente ogni volta che la palla di cannone colpisce il blocco; mostrate il punteggio nell'angolo superiore sinistro dello schermo.
- Ogni volta che una palla di cannone colpisce un obiettivo, inserite l'animazione dell'esplosione di entrambi.
- Aggiungete l'animazione dell'esplosione della palla di cannone ogni volta che colpisce il blocco.
- Quando la palla di cannone colpisce il blocco, aumentate la lunghezza del blocco del 5%.
- Rendete il gioco sempre più difficile aumentando gradualmente la velocità degli obiettivi e del blocco.
- Aumentate il numero dei blocchi che si muovono in maniera indipendente tra il cannone e gli obiettivi.
- Aggiungete un turno "bonus" che dura quattro secondi; per segnalare che si tratta di un bonus, cambiate il colore degli obiettivi e inserite un sottofondo musicale; se l'utente colpisce un obiettivo durante quei quattro secondi, ottiene un bonus di 1.000 punti.
- Consentite all'utente di muovere il cannone verso l'alto e verso il basso mediante i tasti freccia, per poter sparare da posizioni differenti.

10.28 (Introduzione alla data science: visualizzazione dinamica del lancio di una moneta) Modificate la simulazione della legge dei grandi numeri con lancio di un dado dell'Esercizio 10.21, in modo da simulare il lancio di una moneta. Usate la generazione casuale delle cifre 1 e 2 per rappresentare rispettivamente le teste e le croci. Eseguite le simulazioni di 20, 200, 20.000 e 2.000.000 di lanci di moneta. Ottenete approssimativamente il 50% delle volte testa e il 50% delle volte croce? Vedete in azione la "legge dei grandi numeri"?

10.29 (Introduzione alla data science: visualizzazione dinamica del lancio di due dadi) Modificate la simulazione della legge dei grandi numeri con lancio di un dado dell'Esercizio 10.21 in modo da simulare il lancio di due dadi. Calcolate la somma dei due valori. Ogni dado ha un valore da 1 a 6, quindi la somma andrà da 2 a 12, dove 7 è la somma più frequente, mentre 2 e 12 sono le meno frequenti. Il seguente diagramma mostra le 36 possibili combinazioni e le loro corrispettive somme:

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Se lanciate i dadi 36.000 volte:

- i valori 2 e 12 appaiono con una probabilità di 1/36-esimo (2,778%), quindi se ne aspettano circa 1.000 di ognuno;
- i valori 3 e 11 appaiono con una probabilità di 2/36-esimi (5,556%), quindi se ne aspettano circa 2.000 di ognuno, e così via; per i valori 7, se ne aspettano circa 6.000.

Visualizzate un grafico dinamico a barre con una barra per ciascuna somma da 2 a 12 che riepiloghi le loro frequenze. Eseguite la simulazione per 360, 36.000 e 36.000.000 di lanci.

10.30 (*Introduzione alla data science - progetto: visualizzazione dinamica delle statistiche di vincita/perdita del gioco Craps*) Reimplementate la vostra soluzione dell'Esercizio 6.20 usando raylib per creare un grafico a barre dinamico che mostri le vincite e le perdite del primo lancio, del secondo lancio, del terzo lancio e così via. Utilizzate coppie di barre verdi e rosse per indicare rispettivamente le vincite e le perdite per ciascun numero di lanci.

10.31 (*Progetto: gioco “Brick”*) Create un gioco, simile al gioco Cannon, che spari proiettili contro un muro fisso di mattoni. L'obiettivo del gioco è distruggere abbastanza mattoni per riuscire a colpire il bersaglio in movimento che si trova dietro il muro. Più rapidamente si riesce a passare oltre il muro e colpire il bersaglio, maggiore sarà il punteggio ottenuto. Aggiungete diversi strati di muro e un bersaglio mobile piccolo. Segnate il punteggio. Aumentate la difficoltà a ogni turno di gioco, inserendo più strati nel muro, con mattoni più piccoli, e aumentando la velocità di movimento del bersaglio.

10.32 (*Progetto: orologio digitale*) Create un'applicazione che visualizzi un orologio digitale sullo schermo.

10.33 (*Progetto: orologio analogico*) Create un'applicazione che visualizzi un orologio analogico: le lancette delle ore, dei minuti e dei secondi devono essere di lunghezza e spessore appropriati e ruotare con lo scorrere del tempo.

CAPITOLO

11

Sommario del capitolo

- 11.1 Introduzione
- 11.2 File e stream
- 11.3 Creazione di un file ad accesso sequenziale
- 11.4 Lettura di dati da un file ad accesso sequenziale
- 11.5 File ad accesso casuale
- 11.6 Creazione di un file ad accesso casuale
- 11.7 Scrittura di dati in maniera casuale su un file ad accesso casuale
- 11.8 Lettura di dati da un file ad accesso casuale
- 11.9 Caso pratico: sistema per l'elaborazione di transazioni
- 11.10 Programmazione sicura in C
- 11.11 Riepilogo

Elaborazione di file

Obiettivi

- Comprendere i concetti di file e stream.
- Creare e leggere dati da file usando l'elaborazione di file di testo ad accesso sequenziale.
- Creare, aggiornare e leggere dati da file usando l'elaborazione di file ad accesso casuale e file binari.
- Sviluppare un vero e proprio programma di elaborazione di transazioni.
- Studiare la programmazione sicura in C nell'ambito dell'elaborazione dei file.

11.1 Introduzione

Nel Capitolo 1 avete studiato la *gerarchia di dati*. I dati nelle variabili sono *temporanei*: vengono *persi* quando un programma termina. I file vengono usati per la memorizzazione di dati a lungo termine. I computer memorizzano file su dispositivi di memoria secondaria, come dischi allo stato solido, unità flash e dischi rigidi. In questo capitolo spiegheremo come creare, aggiornare ed elaborare file di dati. Considereremo l'elaborazione di file sia ad accesso sequenziale che ad accesso casuale.

11.2 File e stream

Il C vede ogni file semplicemente come uno *stream* (flusso) sequenziale di byte, come mostrato nel seguente diagramma di flusso:



Ogni file termina o con un **marcatore di end-of-file** o dopo uno specifico numero di byte registrato in una struttura di dati amministrativi gestita dal sistema – questo è determinato dalla piattaforma in uso e non si può vedere.

Stream standard in ogni programma

Quando aprite un file in C, a esso viene associato uno **stream**. Quando inizia l'esecuzione di un programma, il C apre automaticamente tre stream:

- lo stream **standard input**, che riceve input dalla tastiera;
- lo stream **standard output**, che stampa output sullo schermo;
- lo stream **standard error**, che stampa messaggi di errore sullo schermo.

Struttura FILE

L'apertura di un file restituisce un puntatore a una **struttura FILE** (definita in `<stdio.h>`) contenente informazioni necessarie al programma per elaborare il file. In alcuni sistemi operativi questa struttura comprende un **descrittore di file**, cioè un indice intero in un array del sistema operativo chiamato **tavella dei file aperti**. Ogni elemento dell'array contiene un **blocco di controllo del file** (*file control block*, FCB) in cui sono memorizzate le informazioni usate dal sistema operativo per amministrare un particolare file. Gli stream di standard input, standard output e standard error vengono manipolati usando i puntatori FILE `stdin`, `stdout` e `stderr`.

Funzione di elaborazione di file `fgetc`

La Libreria Standard fornisce molte funzioni per leggere dati da file e per scrivere dati su file. La funzione `fgetc`, come `getchar`, legge un carattere dal file specificato dal puntatore FILE che riceve come argomento. Per esempio, la chiamata `fgetc(stdin)` legge un carattere dallo stream standard input. Tale chiamata è equivalente alla chiamata `getchar()`.

Funzione di elaborazione di file `fputc`

La funzione `fputc`, come `putchar`, scrive il carattere nel suo primo argomento sul file specificato dal puntatore FILE nel suo secondo argomento. Per esempio, la chiamata di funzione `fputc('a', stdout)` scrive un carattere sullo stream standard output ed è equivalente a `putchar('a')`.

Altre funzioni di elaborazione di file

Diverse altre funzioni usate per leggere dati dallo standard input e scrivere dati sullo standard output hanno funzioni per l'elaborazione di file con nomi simili. Le funzioni `fgets` e `fputs`, per esempio, possono essere usate, rispettivamente, per leggere una riga di testo da un file e per scrivere una riga di testo su un file. Nei prossimi paragrafi introdurremo le funzioni equivalenti a `scanf` e `printf`, ovvero `fscanf` e `fprintf`. Più avanti nel capitolo esamineremo le funzioni `fread` e `fwrite`.

✓ Autovalutazione

1. (*Completare*) Quando inizia l'esecuzione di un programma, il C apre automaticamente tre stream: gli stream _____, _____ e _____.

Risposta: standard input, standard output, standard error.

2. (*Completare*) Il C vede ogni file semplicemente come uno stream sequenziale di byte. Ogni file termina o con un _____ o dopo uno specifico numero di byte registrato in una struttura di dati di amministrativi gestita dal sistema.

Risposta: marcatore di end-of-file.

11.3 Creazione di un file ad accesso sequenziale

Il C non impone alcuna struttura su un file. Pertanto, nozioni come record di un file non fanno parte del linguaggio C. L'esempio seguente mostra come imporre a un file una struttura a record.

La Figura 11.1 crea un semplice file ad accesso sequenziale che potrebbe essere usato in un sistema che gestisce conti che tengono traccia delle somme di denaro dovute (crediti) dai clienti di un'azienda. Per ogni cliente il programma legge il numero di conto, il nome e il saldo del cliente, ossia la somma che il cliente deve all'azienda per beni e servizi ricevuti in passato. I dati ottenuti per ogni cliente costituiscono un "record" per quel cliente. Il numero di conto è la chiave del record di questa applicazione. Questo programma presuppone che l'utente inserisca i record ordinati secondo i numeri di conto. In un grande sistema di gestione di conti, sarebbe invece disponibile una funzionalità che permetterebbe all'utente di inserire i record in un ordine qual-

siasi. Il programma poi metterebbe in ordine i record e li scriverebbe su file. Le Figure 11.2 e 11.3 usano il file di dati creato nella Figura 11.1, pertanto dovete eseguire il programma della Figura 11.1 prima di quelli delle Figure 11.2 e 11.3.

```

1 // fig11_01.c
2 // Creazione di un file sequenziale
3 #include <stdio.h>
4
5 int main(void){
6     FILE *cfPtr = NULL; // cfPtr = puntatore al file clients.txt
7
8     // fopen apre il file. Esce dal programma se non è possibile creare il file
9     if ((cfPtr = fopen("clients.txt", "w")) == NULL) {
10         puts("File could not be opened");
11     }
12     else {
13         puts("Enter the account, name, and balance.");
14         puts("Enter EOF to end input.");
15         printf("%s", "? ");
16
17         int account = 0; // numero del conto
18         char name[30] = ""; // nome del titolare del conto
19         double balance = 0.0; // saldo del conto
20
21         scanf("%d%29s%lf", &account, name, &balance);
22
23         // scrivi numero del conto, nome e saldo nel file con fprintf
24         while (!feof(stdin)) {
25             fprintf(cfPtr, "%d %s %.2f\n", account, name, balance);
26             printf("%s", "? ");
27             scanf("%d%29s%lf", &account, name, &balance);
28         }
29
30         fclose(cfPtr); // fclose chiude il file
31     }
32 }
```

```

Enter the account, name, and balance.
Enter EOF to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Figura 11.1 Creazione di un file sequenziale.

11.3.1 Puntatore a un FILE

La riga 6 definisce cfPtr come un puntatore a una struttura FILE. Un programma fa riferimento a ogni file aperto con un puntatore FILE separato. Per usare i file non è necessario conoscere le caratteristiche specifiche della struttura FILE. Se vi interessa, potete studiarne la dichiarazione in stdio.h.

11.3.2 Usare fopen per aprire il file

La riga 9 chiama `fopen` per creare il file "clients.txt" e stabilisce una "linea di comunicazione" con esso. Il puntatore al file che `fopen` restituisce viene assegnato a `cPtr`.

La funzione `fopen` riceve due argomenti:

- un nome di file (che può includere informazioni sul percorso, per localizzare la posizione del file);
- una **modalità di apertura del file**.

La modalità di apertura del file "w" indica che `fopen` dovrebbe aprire il file per scriverci, ovvero in scrittura.

 Se il file non esiste e la modalità di apertura è "w", `fopen` crea il file. Se apriete un file esistente, `fopen` ne elimina il contenuto *senza avvertimento*. Questo è un errore logico, se non è previsto che il vostro programma sostituisca il file esistente.

L'istruzione `if` determina se il puntatore a file `cPtr` è `NULL`. Se è `NULL`, significa che il file non può essere aperto, probabilmente perché il programma non dispone dell'autorizzazione per creare un file nella cartella specificata. In questo programma, il file viene creato nella stessa cartella del programma. Se `cPtr` è `NULL`, il programma stampa un messaggio di errore e termina. Altrimenti, il programma elabora l'input e lo scrive sul file.

11.3.3 Usare feof per controllare l'indicatore di end-of-file

Il programma richiede all'utente di inserire i vari campi per ogni record oppure di inserire *end-of-file* quando l'inserimento dei dati è completato. Le combinazioni di tasti per inserire un end-of-file sono dipendenti dalla piattaforma:

- Windows: <Ctrl> + z, poi premete *Invio*;
- macOS/Linux: <Ctrl> + d

La riga 24 chiama `feof` per determinare se l'indicatore di end-of-file è impostato per `stdin`. L'indicatore di end-of-file informa il programma che non vi sono più dati da elaborare. Quando l'utente inserisce la *combinazione di tasti di end-of-file*, il sistema operativo imposta l'indicatore di end-of-file per lo stream standard input. L'argomento della funzione `feof` è un puntatore `FILE` al file da testare per l'indicatore di end-of-file (`stdin` in questo caso). La funzione restituisce un valore diverso da zero (*vero*) quando l'indicatore di end-of-file è stato impostato; diversamente, la funzione restituisce zero (*falso*). L'istruzione `while` di questo programma continua l'esecuzione finché l'utente non inserisce l'indicatore di end-of-file.

11.3.4 Usare fprintf per scrivere su un file

La riga 25 scrive un record come riga di testo sul file `clients.txt`. I dati possono essere recuperati in seguito da un programma progettato per leggere il file (Paragrafo 11.4). La funzione `fprintf` è equivalente a `printf`, però `fprintf` riceve come argomento anche un puntatore `FILE` che specifica il file su cui saranno scritti i dati. La funzione `fprintf` può inviare in uscita dati allo standard output usando `stdout` come puntatore a file.

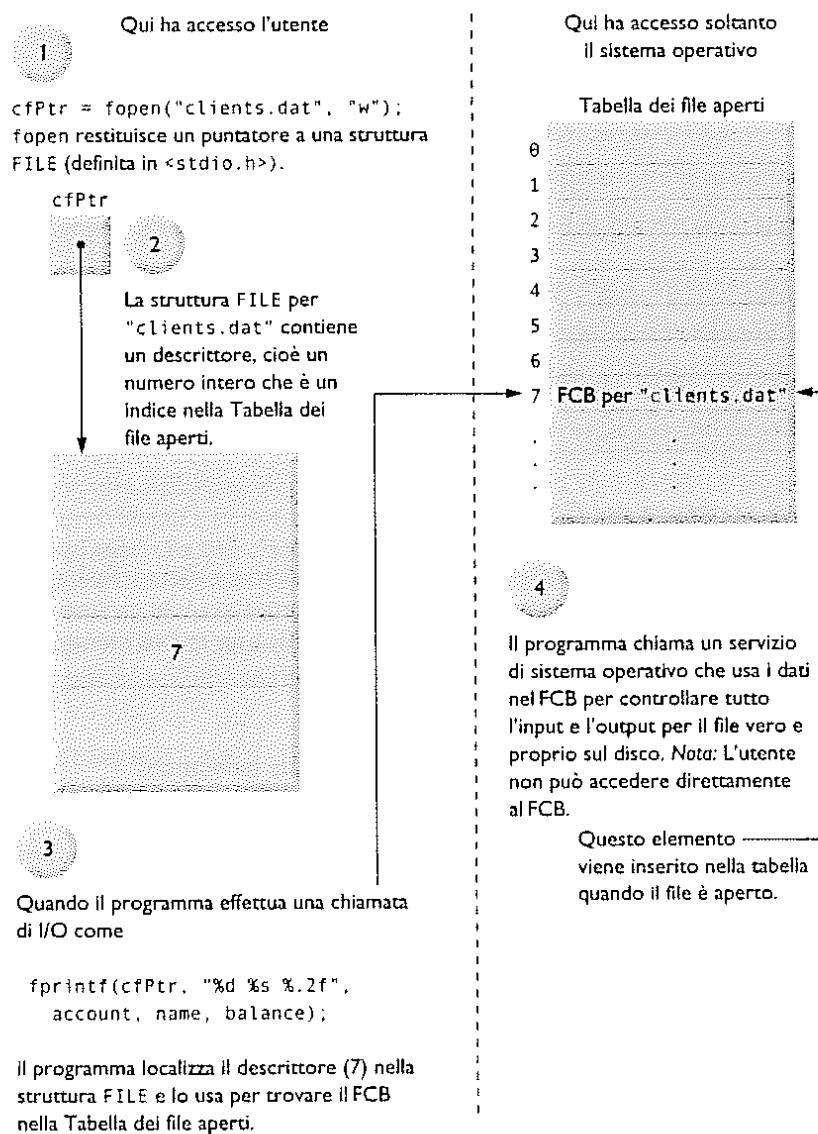
11.3.5 Usare fclose per chiudere il file

Dopo l'inserimento dell'indicatore di end-of-file da parte dell'utente, il programma chiude il file `clients.txt` chiamando `fclose` (riga 30), quindi termina. La funzione `fclose` riceve il puntatore `FILE` come argomento. Se la funzione `fclose` non viene chiamata esplicitamente, il sistema operativo chiuderà normalmente il file al termine dell'esecuzione del programma. Questo è un esempio di "manutenzione" da parte di un sistema operativo. La chiusura di un file può liberare risorse che altri utenti o programmi stanno forse aspettando, perciò dovete chiudere ogni file non appena esso non è più necessario.

Nell'esempio di esecuzione del programma della Figura 11.1, l'utente inserisce informazioni per cinque conti, poi inserisce un end-of-file per segnalare che l'inserimento dei dati è completato. L'esempio di esecuzione non mostra come i record dei dati appaiano realmente nel file. Per verificare che il file è stato creato con successo, nel prossimo paragrafo presenteremo un programma che legge il file e ne stampa il contenuto.

Relazione tra puntatori FILE, strutture FILE e FCB

La seguente figura illustra le relazioni tra i puntatori FILE, le strutture FILE e i FCB. Quando un programma apre il file "clients.txt", il sistema operativo copia in memoria un FCB per il file. La figura mostra la connessione tra il puntatore del file restituito da fopen e il FCB usato dal sistema operativo per amministrare il file. I programmi possono elaborare uno o più file o non elaborare alcun file. Ogni file usato in un programma avrà un differente puntatore a file restituito da fopen. Tutte le successive funzioni di elaborazione di file dopo l'apertura di un file devono riferirsi al file con il puntatore appropriato.



11.3.6 Modalità di apertura dei file

Nella seguente tabella sono riassunte le modalità di apertura dei file. Quelle contenenti la lettera "b" servono per la manipolazione dei file binari, che sono analizzate nei Paragrafi 11.5-11.9 dove introdurremo i file ad accesso casuale.

Modalità	Descrizione
r	Apre un file esistente per la lettura.
w	Crea un file per la scrittura. Se il file esiste già, elimina i contenuti correnti.
a	Apre o crea un file per scrivere alla fine del file – riguarda le operazioni di scrittura che aggiungono dati al file.
r+	Apre un file esistente per l'aggiornamento (lettura e scrittura).
w+	Crea un file per lettura e scrittura. Se il file esiste già, elimina i contenuti correnti.
a+	Apre o crea un file per lettura e aggiornamento dove la scrittura è effettuata alla fine del file – cioè, le operazioni di scrittura aggiungono dati al file.
rb	Apre un file esistente per la lettura in forma binaria.
wb	Crea un file per la scrittura in forma binaria. Se il file esiste già, elimina i contenuti correnti.
ab	Apre o crea un file per la scrittura alla fine del file in forma binaria (<i>append</i>).
rb+	Apre un file esistente per l'aggiornamento (lettura e scrittura) in forma binaria.
wb+	Crea un file per l'aggiornamento in forma binaria. Se il file esiste già, elimina i contenuti correnti.
ab+	Apre o crea un file per l'aggiornamento in forma binaria; la scrittura è effettuata alla fine del file.

Modalità esclusiva di scrittura del C11

Il C11 fornisce una modalità esclusiva di scrittura,¹ indicata con "wx", "w+x", "wbx" o "wb+x". Nella modalità esclusiva di scrittura, fopen fallirà se il file esiste già o non può essere creato. Se l'apertura di un file nella modalità esclusiva di scrittura riesce e il sistema sottostante supporta l'accesso esclusivo al file, allora *solo* il vostro programma può accedere al file finché è aperto. Se si verifica un errore mentre si apre un file in una qualsiasi modalità, fopen restituisce NULL.



Errori comuni di elaborazione del file

Ecco alcuni comuni errori logici di elaborazione dei file che potreste incontrare:

- aprire un file inesistente per la lettura;
- aprire un file per la lettura o la scrittura senza che si siano ottenuti gli opportuni diritti di accesso al file (dipendenti dal sistema operativo);
- aprire un file per scrivere quando non c'è spazio disponibile;
- aprire un file in modalità di scrittura ("w") quando si dovrebbe aprire in modalità di aggiornamento ("r+"); "w" elimina il contenuto del file.

✓ Autovalutazione

1. (*Codice*) Dove viene stampato l'output della seguente istruzione?

```
fprintf(stdout, "%d %s %.2f\n", account, name, balance);
```

Risposta: Sul dispositivo di standard output, che solitamente è lo schermo.

2. (*Vero/Falso*) La nozione di record di un file è integrata nel C.

Risposta: Falso. In realtà, il C non impone alcuna struttura a un file, pertanto nozioni come record di un file non fanno parte del linguaggio C. Si può impostare a un file una struttura a record di propria scelta.

1. Alcuni compilatori e piattaforme non supportano la modalità esclusiva di scrittura.

3. (Completare) Un programma in C amministra ciascun file con una struttura _____ separata.

Risposta: FILE.

4. (Vero/Falso) Quando apriete un file per la scrittura, fopen vi avverte se il file esiste già.

Risposta: Falso. fopen elimina il contenuto del file senza alcun avvertimento.

5. (Scelta multipla) Quale modalità di apertura dei file corrisponde alla descrizione “apre un file esistente per l’aggiornamento (lettura e scrittura)”?

- a) u.
- b) rw.
- c) r+.
- d) w+.

Risposta: c.

11.4 Lettura di dati da un file ad accesso sequenziale

I dati vengono memorizzati in un file in modo che, quando servono, si possano recuperare per l’elaborazione. Il paragrafo precedente ha illustrato come creare un file per l’accesso sequenziale. Il presente paragrafo mostra come leggere dati da un file in maniera sequenziale. Se i contenuti di un file non devono essere modificati, aprirete il file solo per la lettura. Ciò previene la modifica non intenzionale dei contenuti del file ed è un altro esempio del principio del privilegio minimo.

Il programma della Figura 11.2 legge e stampa record del file "clients.txt" creato nella Figura 11.1. La riga 6 definisce il puntatore FILE cfPtr. La riga 9 tenta di aprire il file per la lettura ("r") e determina se si è aperto con successo (vale a dire che fopen non ha restituito NULL). La riga 18 legge un "record" dal file. La funzione fscanf è equivalente a scanf ma riceve come suo primo argomento un puntatore FILE al file da cui sono letti i dati. La prima volta che questa istruzione viene eseguita, account avrà il valore 100, name il valore "Jones" e balance il valore 24.98. Ogni successiva chiamata a fscanf (riga 23), il programma legge un altro record dal file e assegna nuovi valori ad account, name e balance. Quando non ci sono più dati da leggere, la riga 26 chiude il file e il programma termina. La funzione feof restituisce *vero* solo dopo che il programma tenta di leggere oltre l’ultima riga del file.

```

1 // fig11_02.c
2 // Lettura e stampa di un file sequenziale
3 #include <stdio.h>
4
5 int main(void) {
6     FILE *cfPtr = NULL; // cfPtr = puntatore al file clients.txt
7
8     // fopen apre il file; esce dal programma se il file non puo' essere aperto
9     if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
10         puts("File could not be opened");
11     }
12     else { // leggi account, name e balance dal file
13         int account = 0; // numero del conto
14         char name[30] = ""; // nome del titolare del conto
15         double balance = 0.0; // saldo del conto
16
17         printf("%-10s%-13s%5s\n", "Account", "Name", "Balance");
18         fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
19
20         // finche' non si incontra un end-of-file
21         while (!feof(cfPtr)) {
22             printf("%-10d%-13s%7.2f\n", account, name, balance);

```

```

23         fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
24     }
25
26     fclose(cfPtr); // fclose chiude il file
27 }
28 }
```

Account	Name	Balance
100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	-42.16
500	Rich	224.62

Figura 11.2 Lettura e stampa di un file sequenziale.

11.4.1 Reimpostare il puntatore di posizione del file

Per recuperare dati da un file in maniera sequenziale, un programma normalmente legge dall'inizio del file e prosegue finché non vengono trovati i dati desiderati. In alcuni casi, un programma deve elaborare un file in maniera sequenziale diverse volte dall'inizio. L'istruzione

```
rewind(cfPtr);
```

riposiziona il **puntatore di posizione del file** all'inizio (byte 0) del file puntato da cfPtr. Il puntatore di posizione del file non è realmente un puntatore. Piuttosto, è un valore intero che indica il numero di byte del successivo byte da leggere o scrivere. Questo è a volte chiamato **offset di file**. Il puntatore di posizione del file è un membro della struttura FILE associata a ogni file.

11.4.2 Programma di interrogazione per il credito

Il programma della Figura 11.3 permette a un gestore del credito di ottenere liste di clienti con:

- saldo zero (ossia clienti non debitori);
- clienti con saldo a credito (ossia clienti ai quali l'azienda deve del denaro);
- clienti con saldo a debito (ossia clienti che devono denaro all'azienda per beni e servizi ricevuti).

Un saldo a credito è una somma negativa; un saldo a debito è una somma positiva. Il programma stampa un menu e permette al gestore del credito di inserire una delle seguenti quattro opzioni:

- l'opzione 1 produce una lista di conti con saldi zero;
- l'opzione 2 produce una lista di conti con saldi a credito;
- l'opzione 3 produce una lista di conti con saldi a debito;
- l'opzione 4 termina l'esecuzione del programma.

```

1 // fig11_03.c
2 // Programma di interrogazione per il credito
3 #include <stdbool.h>
4 #include <stdio.h>
5
6 enum Options {ZERO_BALANCE = 1, CREDIT_BALANCE, DEBIT_BALANCE, END};
7
8 // determina se stampare un record
9 bool shouldDisplay(enum Options option, double balance) {
```

```
10     if ((option == ZERO_BALANCE) && (balance == 0)) {
11         return true;
12     }
13
14     if ((option == CREDIT_BALANCE) && (balance < 0)) {
15         return true;
16     }
17
18     if ((option == DEBIT_BALANCE) && (balance > 0)) {
19         return true;
20     }
21
22     return false;
23 }
24
25 int main(void) {
26     FILE *cfPtr = NULL; // puntatore al file clients.txt
27
28     // fopen apre il file; esce dal programma se il file non puo' essere aperto
29     if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
30         puts("File could not be opened");
31     }
32     else {
33         // stampa le opzioni di richiesta
34         printf("%s", "Enter request\n"
35                 " 1 - List accounts with zero balances\n"
36                 " 2 - List accounts with credit balances\n"
37                 " 3 - List accounts with debit balances\n"
38                 " 4 - End of run\n? ");
39         int request = 0;
40         scanf("%d", &request);
41
42         // stampa record
43         while (request != END) {
44             switch (request) {
45                 case ZERO_BALANCE:
46                     puts("\nAccounts with zero balances:");
47                     break;
48                 case CREDIT_BALANCE:
49                     puts("\nAccounts with credit balances:");
50                     break;
51                 case DEBIT_BALANCE:
52                     puts("\nAccounts with debit balances:");
53                     break;
54             }
55
56             int account = 0;
57             char name[30] = "";
58             double balance = 0.0;
59
60             // lettura di numero del conto, nome e saldo dal file
61             fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
```

```

62
63     // lettura dei contenuti del file (fino a eof)
64     while (!feof(cfPtr)) {
65         // scrive solo se balance è 0
66         if (shouldDisplay(request, balance)) {
67             printf("%-10d%-13s%7.2f\n", account, name, balance);
68         }
69
70         // lettura di numero del conto, nome e saldo dal file
71         fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
72     }
73
74     rewind(cfPtr); // riporta cfPtr all'inizio del file
75
76     printf("%s", "\n? ");
77     scanf("%d", &request);
78 }
79
80 puts("End of run.");
81 fclose(cfPtr); // chiude il file
82 }
83 }
```

```

Enter request
1 - List accounts with zero balances
2 - List accounts with credit balances
3 - List accounts with debit balances
4 - End of run
? 1

Accounts with zero balances:
300      White      0.00

? 2

Accounts with credit balances:
400      Stone     -42.16

? 3

Accounts with debit balances:
100      Jones     24.98
200      Doe       345.67
500      Rich      224.62

? 4
End of run.
```

Figura 11.3 Programma di interrogazione per il credito.

Aggiornamento di un file sequenziale

I dati in questo tipo di file sequenziale non possono essere modificati senza il rischio di distruggere altri dati. Per esempio, se il nome "White" deve essere cambiato in "Worthington", non è sufficiente sovrascrivere il vecchio nome. Il record per White è stato scritto sul file come

```
300 White 0.00
```

Se dovete riscrivere il record iniziando dalla stessa posizione nel file utilizzando il nuovo nome, il record sarebbe

```
300 Worthington 0.00
```

Il nuovo record ha più caratteri del record originario. I caratteri oltre la seconda "o" in "Worthington" sovrascriveranno l'inizio del record sequenziale successivo nel file. Il problema qui è che nel **modello di input/output formattato** con cui vengono usate `fprintf` e `fscanf` i campi e i record possono variare in dimensioni. Per esempio, i valori 7, 14, -117, 2074 e 27383 sono tutti `int` memorizzati internamente nello stesso numero di byte, ma richiedono campi di dimensioni differenti quando sono stampati sullo schermo o scritti su un file come testo.

Pertanto, l'accesso sequenziale con `fprintf` e `fscanf` solitamente non viene usato per aggiornare i record sul posto. Invece, l'intero file viene riscritto. In un file ad accesso sequenziale, per effettuare il cambio di nome precedente è necessario

- copiare i record che vengono prima di `300 White 0.00` in un nuovo file;
- scrivere il nuovo record;
- copiare i record dopo `300 White 0.00` nel nuovo file;
- sostituire il vecchio file con quello nuovo.

Questo richiede l'elaborazione di ogni record nel file per aggiornare un record.

✓ Autovalutazione

1. (*Completare*) La funzione `fscanf` è equivalente alla funzione `scanf`, ma `fscanf` riceve come argomento un _____.

Risposta: puntatore a file per il file dal quale leggere i dati.

2. (*Vero/Falso*) La funzione `feof` restituisce *vero* solo dopo che il programma tenta di leggere i dati inesistenti dopo l'ultima riga.

Risposta: *Vero*.

3. (*Completare*) La seguente istruzione riposiziona un _____ di un file al byte 0 del file.

```
rewind(cfPtr);
```

Risposta: puntatore di posizione.

4. (*Vero/Falso*) Nel modello di input/output formattato con cui vengono usate le funzioni `fprintf` e `fscanf`, i campi (e di conseguenza i record) sono di dimensioni fisse.

Risposta: *Falso*. In realtà, in questo modello, i campi (e di conseguenza i record) possono variare in dimensioni.

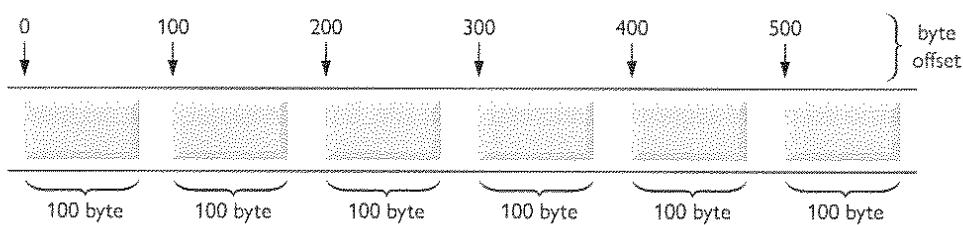
11.5 File ad accesso casuale

I record in un file creato con la funzione di output formattato `fprintf` non sono necessariamente della stessa lunghezza. Invece, il **file ad accesso casuale** utilizza record di lunghezza fissa accessibili direttamente (e quindi velocemente) senza effettuare ricerche in altri record. Questo rende i file ad accesso casuale adatti per i **sistemi di elaborazione di transazioni** che richiedono l'accesso rapido a dati specifici, come i sistemi di prenotazione delle compagnie aeree, i sistemi bancari e i sistemi informatici dei punti vendita. Vi sono altri

modi per implementare file ad accesso casuale, ma limiteremo la nostra analisi a questo semplice approccio, usando record di lunghezza fissa.

Poiché ogni record in un file ad accesso casuale ha normalmente la stessa lunghezza, la posizione esatta di un record rispetto all'inizio del file può essere calcolata come una funzione della chiave del record. Vedremo presto come ciò facilita l'accesso immediato a record specifici anche in file di grandi dimensioni.

La seguente figura illustra un modo per implementare un file ad accesso casuale. Un tale file è come un treno merci con molti vagoni, alcuni vuoti e alcuni con un carico. Ogni vagone del treno ha la stessa lunghezza.



I record di lunghezza fissa permettono a un programma di inserire dati in un file ad accesso casuale *senza distruggere altri dati del file*. È anche possibile aggiornare o cancellare i dati memorizzati in precedenza senza riscrivere l'intero file. Nei paragrafi seguenti spiegheremo come

- creare un file ad accesso casuale;
- inserire i dati;
- leggere i dati sia in modo sequenziale che in modo casuale;
- aggiornare i dati;
- cancellare quelli non più necessari.

✓ Autovalutazione

1. (*Vero/Falso*) Si può accedere direttamente, senza effettuare ricerche in altri record, ai singoli record che vengono scritti e letti da un file ad accesso casuale. Questo rende i file ad accesso casuale adatti per i sistemi che richiedono un accesso rapido a dati specifici.

Risposta: Vero.

2. (*Completare*) I record di un file ad accesso casuale hanno tutti la stessa lunghezza, pertanto la posizione esatta di un record rispetto all'inizio del file può essere calcolata in base alla _____.

Risposta: chiave del record.

11.6 Creazione di un file ad accesso casuale

La funzione `fwrite` trasferisce su un file un numero specificato di byte a cominciare da una data posizione in memoria. I dati sono scritti a iniziare dalla posizione corrente nel file indicata dal puntatore di posizione. La funzione `fread` trasferisce un numero specificato di byte dalla posizione corrente nel file indicata dal puntatore di posizione a una determinata area nella memoria. Quando scriviamo un intero di quattro byte con

```
fprintf(fPtr, "%d", number);
```

l'output potrebbe includere fino a 11 cifre (10 cifre più un segno; ogni cifra richiede almeno un byte di memoria). Con i file ad accesso casuale, l'istruzione

```
fwrite(&number, sizeof(int), 1, fPtr);
```

scrive *sempre* quattro byte (su un sistema con interi di quattro byte) contenuti in una variabile `number` di tipo `int` sul file rappresentato da `fPtr`. Descriveremo a breve l'argomento 1. In seguito, possiamo usare `fread` per leggere quei quattro byte in una variabile `int`. Sebbene `fread` e `fwrite` leggano e scrivano dati in una

dimensione fissa invece che in un formato di dimensione variabile, li elaborano come byte “raw”, invece che nel formato testo di `printf` e `scanf` facilmente leggibile dagli esseri umani. Poiché la rappresentazione “raw” (letteralmente “grezza”) dei dati è dipendente dal sistema, i dati “raw” potrebbero non essere leggibili su altri sistemi, o da parte di programmi prodotti da altri compilatori oppure con altre opzioni di compilazione.

`fwrite` e `fread` possono scrivere e leggere array

Le funzioni `fwrite` e `fread` possono leggere e scrivere array. Il terzo argomento sia di `fwrite` che di `fread` è il numero di elementi dell’array che deve essere scritto o letto. La precedente chiamata alla funzione `fwrite` scrive un singolo intero su un file, cosicché il terzo argomento è 1 (come se si scrivesse un solo elemento di un array). I programmi di elaborazione di file raramente scrivono un singolo campo su un file. Normalmente scrivono un elemento `struct` alla volta, come mostreremo negli esempi seguenti.

Problema

Considerate il seguente problema.

Create un sistema di elaborazione di transazioni in grado di memorizzare fino a 100 record di lunghezza fissa. Ogni record deve avere un numero di conto che sarà usato come chiave del record, un cognome, un nome e un saldo. Il programma deve usare un file ad accesso casuale ed essere in grado di aggiornare un conto, inserire un nuovo conto, cancellare un conto ed elencare tutti i record in un file di testo formattato per la stampa.

I paragrafi successivi introdurranno le tecniche necessarie per creare il programma di elaborazione di transazioni. Il programma della Figura 11.4 mostra come aprire un file ad accesso casuale, definire un formato di record usando il costrutto `struct`, scrivere dati nel file e chiudere il file. Questo programma inizializza tutti e 100 i record del file “accounts.dat” con elementi `struct` vuoti usando la funzione `fwrite`. Ogni elemento `struct` vuoto contiene il numero di conto 0, stringhe vuote (“ ”) per il cognome e il nome, e il saldo 0.0. Inizializziamo tutti i record per creare lo spazio nel quale il file sarà memorizzato e perché sia possibile determinare se un record contiene dati.

```

1 // fig11_04.c
2 // Creazione di un file ad accesso casuale in maniera sequenziale
3 #include <stdio.h>
4
5 // definizione della struttura clientData
6 struct clientData {
7     int account;
8     char lastName[15];
9     char firstName[10];
10    double balance;
11 };
12
13 int main(void) {
14     FILE *cfPtr = NULL; // puntatore al file accounts.dat
15
16     // fopen apre il file; esce se il file non puo' essere aperto
17     if ((cfPtr = fopen("accounts.dat", "wb")) == NULL) {
18         puts("File could not be opened.");
19     }
20     else {
21         // crea clientData con informazioni predefinite
22         struct clientData blankClient = {0, "", "", 0.0};
23
24         // scrive 100 record vuoti su file
25         for (int i = 1; i <= 100; ++i) {

```

```

26     fwrite(&blankClient, sizeof(struct clientData), 1, cfPtr);
27 }
28
29 fclose (cfPtr); // fclose chiude il file
30 }
31 }
```

Figura 11.4 Creazione di un file ad accesso casuale in maniera sequenziale.

La riga 17 apre il file "accounts.dat" per la scrittura binaria ("wb"). La funzione `fwrite` (riga 26) scrive un blocco di byte su un file. Gli argomenti sono:

- `&blankClient`: l'indirizzo dell'oggetto da scrivere;
- `sizeof(struct clientData)`: la dimensione in byte dell'oggetto da scrivere;
- 1: il numero degli oggetti di quella dimensione da scrivere;
- `cfPtr`: un `FILE *` che rappresenta il file nel quale i byte vengono memorizzati.

Ricordate che `sizeof` restituisce la dimensione in byte del suo operando, `struct clientData`.

Scrivere un array di oggetti

Alla riga 26, la funzione `fwrite` scrive un oggetto che non è un elemento di un array. Per scrivere un array, passatelo a `fwrite` come primo argomento e specificate come terzo argomento il numero di elementi da stampare.

✓ Autovalutazione

1. (*Vero/Falso*) Per una variabile `number` con interi di quattro byte, l'istruzione seguente scrive sempre quattro byte, anche se la rappresentazione testuale di `number` potrebbe arrivare fino a 11 cifre:

```
fwrite(&number, sizeof(int), 1, fPtr);
```

Risposta: *Vero*.

2. (*Completare*) Le funzioni `fread` e `fwrite` leggono e scrivono dati nel formato "raw", ovvero come _____ di dati.

Risposta: byte.

3. (*Completare*) La funzione `fwrite` può scrivere più elementi di un array. Nella chiamata a `fwrite`, specificate un puntatore a un array e _____ rispettivamente come primo e terzo argomento.

Risposta: il numero di elementi da scrivere.

11.7 Scrittura di dati in maniera casuale su un file ad accesso casuale

[*Nota*: Le Figure 11.5, 11.6 e 11.7 usano il file di dati creato nella Figura 11.4, quindi dovete eseguire il programma della Figura 11.4 prima di quelli delle Figure 11.5, 11.6 e 11.7.]

Il programma della Figura 11.5 scrive dati sul file "accounts.dat". Esso utilizza `fseek` e `fwrite` per memorizzare dati in posizioni specifiche del file. La funzione `fseek` imposta il puntatore di posizione del file a una posizione specifica in termini di byte, quindi `fwrite` scrive i dati lì.

```

1 // fig11_05.c
2 // Scrittura di dati in maniera casuale su un file ad accesso casuale
3 #include <stdio.h>
4
5 // definizione della struttura clientData
6 struct clientData {
```

```

7   int account;
8   char lastName[15];
9   char firstName[10];
10  double balance;
11 } // fine della struttura clientData
12
13 int main(void) {
14     FILE *cfPtr = NULL; // puntatore al file accounts.dat
15
16     // fopen apre il file; esce se il file non puo' essere aperto
17     if ((cfPtr = fopen("accounts.dat", "rb+")) == NULL) {
18         puts("File could not be opened.");
19     }
20     else {
21         // crea un oggetto clientData con informazioni predefinite
22         struct clientData client = {0, "", "", 0.0};
23
24         // richiedi all'utente di specificare il numero di conto
25         printf("%s", "Enter account number (1 to 100, 0 to end input): ");
26         scanf("%d", &client.account);
27
28         // l'utente inserisce le informazioni, che vengono copiate sul file
29         while (client.account != 0) {
30             // l'utente inserisce il cognome, il nome e il saldo
31             printf("%s", "Enter lastname, firstname, balance: ");
32
33             // imposta il record con il cognome, il nome e il valore del saldo
34             fscanf(stdin, "%14s%9s%lf", client.lastName,
35                   client.firstName, &client.balance);
36
37             // cerca (seek) nel file la posizione del record specificato
38             fseek(cfPtr, (client.account - 1) *
39                   sizeof(struct clientData), SEEK_SET);
39
40             // scrivi le informazioni specificate dall'utente nel file
41             fwrite(&client, sizeof(struct clientData), 1, cfPtr);
42
43             // consenti all'utente di inserire un altro numero di conto
44             printf("%s", "\nEnter account number: ");
45             scanf("%d", &client.account);
46         }
47     }
48
49     fclose(cfPtr); // fclose chiude il file
50 }
51 }
```

```

Enter account number (1 to 100, 0 to end input): 37
Enter lastname, firstname, balance: Barker Doug 0.00

Enter account number: 29
Enter lastname, firstname, balance: Brown Nancy -24.54

Enter account number: 96
Enter lastname, firstname, balance: Stone Sam 34.98
```

```

Enter account number: 88
Enter lastname, firstname, balance: Smith Dave 258.34
Enter account number: 33
Enter lastname, firstname, balance: Dunn Stacey 314.33
Enter account number: 0

```

Figura 11.5 Scrittura di dati in maniera casuale su un file ad accesso casuale.

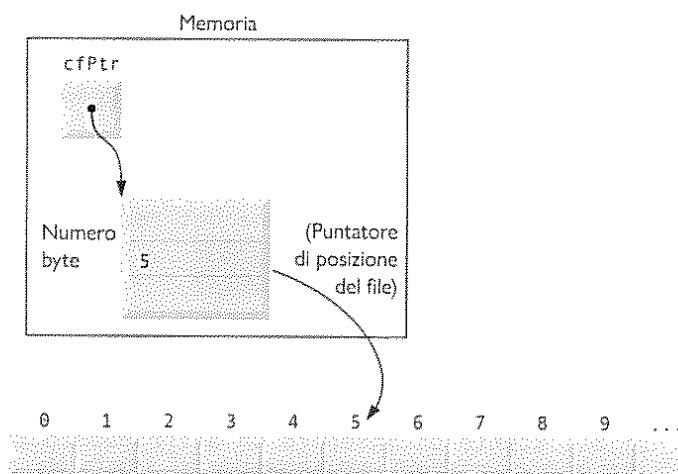
11.7.1 Collocare il puntatore di posizione del file con fseek

Le righe 38-39 collocano il puntatore di posizione per il file a cui cfPtr fa riferimento nel punto in termini di byte calcolato da

```
(client.account - 1) * sizeof(struct clientData)
```

Il valore di questa espressione è chiamato **offset o spostamento**. In questo esempio, il numero di conto è compreso nell'intervallo 1-100. Il file inizia con il byte 0, quindi viene sottratto 1 dal numero di conto quando si calcola la posizione del record riferita ai byte. Per il record 1, le righe 38-39 impostano il puntatore di posizione del file al byte 0 del file. La costante simbolica SEEK_SET indica che fseek deve spostare il puntatore di posizione del file in riferimento all'inizio del file.

Il diagramma seguente mostra il puntatore FILE che si riferisce a una struttura FILE in memoria. Il puntatore di posizione del file in questo diagramma indica che il byte successivo da leggere o da scrivere è il numero 5.



Prototipo di funzione per fseek

Il prototipo di funzione per fseek è

```
int fseek(FILE *stream, long int offset, int whence);
```

dove offset è il numero di byte di cui ci si deve spostare rispetto a whence nel file puntato da stream. Un offset positivo indica uno spostamento in avanti e uno negativo uno spostamento all'indietro. L'argomento whence può essere SEEK_SET, SEEK_CUR o SEEK_END (tutti definiti in <stdio.h>), che indicano la posizione di riferimento per il posizionamento:

- SEEK_SET indica l'*inizio* del file.
- SEEK_CUR indica la *posizione corrente* nel file.
- SEEK_END indica la *fine* del file.

Dovreste usare solo offset positivi con SEEK_SET e solo quelli negativi con SEEK_END.

11.7.2 Controllo sugli errori

Per semplicità, i programmi di questo capitolo *non* eseguono alcun controllo sugli errori. I programmi a livello industriale devono determinare se funzioni come `fscanf` (Figura 11.5, righe 34-35), `fseek` (righe 38-39) e `fwrite` (riga 42) operano correttamente controllando i loro valori di ritorno. La funzione `fscanf` restituisce il numero di dati letti con successo o il valore `EOF` se si verifica un problema durante la lettura dei dati. La funzione `fseek` restituisce un valore diverso da zero se l'operazione di accesso non può essere eseguita (per esempio, il tentativo di accesso a una posizione prima dell'avvio del file). La funzione `fwrite` restituisce il numero di elementi che ha inviato in uscita con successo. Se questo numero è minore del *terzo argomento* nella chiamata della funzione, si è verificato un errore di scrittura.

✓ Autovalutazione

1. (*Completare*) La funzione _____ imposta il puntatore di posizione del file a una posizione specifica in termini di byte.

Risposta: `fseek`.

2. (*Completare*) La costante simbolica _____ indica che il puntatore di posizione del file deve essere posizionato in riferimento all'inizio del file.

Risposta: `SEEK_SET`.

11.8 Lettura di dati da un file ad accesso casuale

La funzione `fread` legge un numero specificato di byte da un file in memoria. Per esempio,

```
fread(&client, sizeof(struct clientData), 1, cfPtr);
```

legge il numero di byte determinato da `sizeof(struct clientData)` dal file a cui `cfPtr` fa riferimento, memorizza i dati in `client` e restituisce il numero di elementi letti. I byte vengono letti dalla posizione specificata dal puntatore di posizione del file.

La funzione `fread` può leggere più elementi di un array di dimensione fissa fornendo un puntatore all'array nel quale memorizzare gli elementi e indicando il numero di elementi da leggere. L'istruzione precedente legge un solo elemento. Per leggerne più di uno, specificate il numero di elementi come terzo argomento di `fread`. La funzione `fread` restituisce il numero di elementi letti con successo. Se questo numero è minore del terzo argomento nella chiamata della funzione, allora si è verificato un errore di lettura.

Il programma della Figura 11.6 legge in maniera sequenziale ogni record nel file "accounts.dat", determina se ogni record contiene dati e, in caso affermativo, stampa i dati formattati. La funzione `feof` determina quando viene raggiunta la fine del file, e la funzione `fread` (righe 28-29) trasferisce dati dal file alla struttura `clientData` `client`.

```
1 // fig11_06.c
2 // Lettura sequenziale di un file ad accesso casuale
3 #include <stdio.h>
4
5 // definizione della struttura clientData
6 struct clientData {
7     int account;
8     char lastName[15];
9     char firstName[10];
10    double balance;
11 };
12
13 int main(void){
14     FILE *cfPtr = NULL; // puntatore al file accounts.dat
15
16     // fopen apre il file; esce se il file non puo' essere aperto
```

```

17     if ((cfPtr = fopen("accounts.dat", "rb")) == NULL) {
18         puts("File could not be opened.");
19     }
20     else {
21         printf("%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
22               "First Name", "Balance");
23
24         // leggi tutti i record dal file (fino a eof)
25         while (!feof(cfPtr)) {
26             // leggi un record
27             struct clientData client = {0, "", "", 0.0};
28             size_t result =
29                 fread(&client, sizeof(struct clientData), 1, cfPtr);
30
31             // stampa il record
32             if (result != 0 && client.account != 0) {
33                 printf("%-6d%-16s%-11s%10.2f\n", client.account,
34                       client.lastName, client.firstName, client.balance);
35             }
36         }
37
38         fclose(cfPtr); // fclose chiude il file
39     }
40 }
```

Acct	Last Name	First Name	Balance
29	Brown	Nancy	-24.54
33	Dunn	Stacey	314.33
37	Barker	Doug	0.00
88	Smith	Dave	258.34
96	Stone	Sam	34.98

Figura 11.6 Lettura sequenziale di un file ad accesso casuale.

✓ Autovalutazione

1. (*Vero/Falso*) La funzione `fread` restituisce il numero di byte che ha letto con successo.

Risposta: *Falso*. La funzione `fread` restituisce il numero di *elementi* che ha letto con successo. Ciascun elemento può essere costituito da più byte. Se il numero di elementi è minore del terzo argomento di `fread`, allora si è verificato un errore di lettura.

2. (*Vero/Falso*) La funzione `fread` può leggere più elementi di un array di dimensione fissa fornendo un puntatore all'array nel quale memorizzare gli elementi e indicando il numero di elementi da leggere.

Risposta: *Vero*.

11.9 Caso pratico: sistema per l'elaborazione di transazioni

Ora creiamo un programma per l'elaborazione di transazioni (Figura 11.7) usando file ad accesso casuale. Il programma gestisce le informazioni relative ai conti correnti di una banca, aggiornando i conti esistenti, aggiungendo nuovi conti, cancellando conti e memorizzando un elenco di tutti i conti correnti in un file di testo per la stampa. Supponiamo che il programma della Figura 11.4 abbia creato il file `accounts.dat`.

Opzione 1: creare un elenco formattato di account

Il programma ha cinque opzioni; l'opzione 5 termina il programma. L'opzione 1 chiama la funzione `textFile` (righe 58-86) per memorizzare un report formattato di tutti i conti in un file di testo chiamato `accounts.txt`, che può essere stampato in seguito. La funzione usa `fread` e le tecniche sequenziali di accesso a file mostrate nella Figura 11.6. Dopo l'opzione 1, `accounts.txt` contiene:

Acct	Last Name	First Name	Balance
29	Brown	Nancy	-24.54
33	Dunn	Stacey	314.33
37	Barker	Doug	0.00
88	Smith	Dave	258.34
96	Stone	Sam	34.98

Opzione 2: aggiornare un account

L'opzione 2 chiama la funzione `updateRecord` (righe 89-125) per aggiornare un conto. La funzione aggiorna solo un record che esiste già, per cui la funzione dapprima controlla se il record specificato dall'utente è vuoto. Come prima cosa, leggiamo il record nella struttura `client` con `fread`. Se il membro `account` è 0, il record non contiene informazioni. Quindi, il programma stampa un messaggio che dice che il record è vuoto, poi ristampa le scelte del menu. Se il record contiene informazioni, la funzione `updateRecord` legge in ingresso l'ammontare della transazione, calcola il nuovo saldo e riscrive il record sul file. Un output tipico per l'opzione 2 è:

```
Enter account to update (1 - 100): 37
37    Barker          Doug      0.00

Enter charge (+) or payment (-): +87.99
37    Barker          Doug     87.99
```

Opzione 3: creare un nuovo account

L'opzione 3 chiama la funzione `newRecord` (righe 128-161) per aggiungere un nuovo conto al file. Se l'utente inserisce un numero di conto di un conto esistente, `newRecord` stampa un messaggio di errore che indica che il record contiene già informazioni e vengono quindi ristampate le scelte del menu. Questa funzione usa lo stesso processo del programma nella Figura 11.5 per aggiungere un nuovo conto. Un output tipico per l'opzione 3 è:

```
Enter new account number (1 - 100): 22
Enter lastname, firstname, balance?
? Johnston Sarah 247.45
```

Opzione 4: eliminare un account

L'opzione 4 chiama la funzione `deleteRecord` (righe 164-190) per cancellare un record dal file. La cancellazione è effettuata chiedendo all'utente il numero del conto e reinizializzando il record. Se il conto non contiene informazioni, `deleteRecord` stampa un messaggio di errore che indica che il conto non esiste.

Codice del programma per l'elaborazione di transazioni

Il programma è mostrato nella Figura 11.7. Il file "accounts.dat" viene aperto per l'aggiornamento (lettura e scrittura) usando la modalità "rb+".

```
1 // fig11_07.c
2 // Il programma per l'elaborazione di transazioni legge sequenzialmente
3 // un file ad accesso casuale, aggiorna i dati già scritti sul file,
4 // crea nuovi dati da inserire nel file e cancella dati dal file.
5 #include <stdio.h>
```

```
6
7 // definizione della struttura clientData
8 struct clientData {
9     int account;
10    char lastName[15];
11    char firstName[10];
12    double balance;
13 };
14
15 // prototipi
16 int enterChoice(void);
17 void textFile(FILE *readPtr);
18 void updateRecord(FILE *fPtr);
19 void newRecord(FILE *fPtr);
20 void deleteRecord(FILE *fPtr);
21
22 int main(void) {
23     FILE *cfPtr = NULL; // puntatore al file accounts.dat
24
25     // fopen apre il file; esce se il file non puo' essere aperto
26     if ((cfPtr = fopen("accounts.dat", "rb+")) == NULL) {
27         puts("File could not be opened.");
28     }
29     else {
30         int choice = 0; // utente
31
32         // consente all'utente di specificare l'azione
33         while ((choice = enterChoice()) != 5) {
34             switch (choice) {
35                 case 1: // crea un file di testo dal file di record
36                     textFile(cfPtr);
37                     break;
38                 case 2: // aggiorna un record
39                     updateRecord(cfPtr);
40                     break;
41                 case 3: // crea un record
42                     newRecord(cfPtr);
43                     break;
44                 case 4: // cancella un record esistente
45                     deleteRecord(cfPtr);
46                     break;
47                 default: // stampa un messaggio per una scelta non valida
48                     puts("Incorrect choice");
49                     break;
50             }
51         }
52
53         fclose(cfPtr); // fclose chiude il file
54     }
55 }
56
57 // crea un file di testo formattato per la stampa
58 void textFile(FILE *readPtr) {
```

```

59     FILE *writePtr = NULL; // puntatore al file accounts.txt
60
61     // fopen apre il file; esce se il file non puo' essere aperto
62     if ((writePtr = fopen("accounts.txt", "w")) == NULL) {
63         puts("File could not be opened.");
64     }
65     else {
66         rewind(readPtr); // sposta il puntatore all'inizio del file
67         fprintf(writePtr, "%-6s%-16s%-11s%10s\n",
68                 "Acct", "Last Name", "First Name", "Balance");
69
70         // copia tutti i record dal file ad accesso casuale nel file di testo
71         while (!feof(readPtr)) {
72             // crea clientData con informazioni predefinite
73             struct clientData client = {0, "", "", 0.0};
74             size_t result =
75                 fread(&client, sizeof(struct clientData), 1, readPtr);
76
77             // scrive un singolo record sul file di testo
78             if (result != 0 && client.account != 0) {
79                 fprintf(writePtr, "%-6d%-16s%-11s%10.2f\n", client.account,
80                         client.lastName, client.firstName, client.balance);
81             }
82         }
83
84         fclose(writePtr); // fclose chiude il file
85     }
86 }
87
88 // aggiorna il saldo nel record
89 void updateRecord(FILE *fPtr) {
90     // ottieni il numero di conto da aggiornare
91     printf("%s", "Enter account to update (1 - 100): ");
92     int account = 0; // numero del conto
93     scanf("%d", &account);
94
95     // sposta il puntatore del file al record corretto nel file
96     fseek(fPtr, (account - 1) * sizeof(struct clientData), SEEK_SET);
97
98     // leggi il record dal file
99     struct clientData client = {0, "", "", 0.0};
100    fread(&client, sizeof(struct clientData), 1, fPtr);
101
102    // stampa un messaggio di errore se il conto non esiste
103    if (client.account == 0) {
104        printf("Account #%d has no information.\n", account);
105    }
106    else { // aggiorna il record
107        printf("%-6d%-16s%-11s%10.2f\n\n", client.account, client.lastName,
108               client.firstName, client.balance);
109
110        // richiedi l'ammontare della transazione all'utente
111        printf("%s", "Enter charge (+) or payment (-): ");

```

```
112     double transaction = 0.0; // ammontare della transazione
113     scanf("%lf", &transaction);
114     client.balance += transaction; // aggiorna il saldo del record
115
116     printf("%-6d%-16s%-11s%10.2f\n", client.account, client.lastName,
117           client.firstName, client.balance);
118
119     // sposta il puntatore del file al record corretto nel file
120     fseek(fPtr, (account - 1) * sizeof(struct clientData), SEEK_SET);
121
122     // scrivi il record aggiornato al posto del vecchio record nel file
123     fwrite(&client, sizeof(struct clientData), 1, fPtr);
124 }
125 }
126
127 // crea e inserisci un record
128 void newRecord(FILE *fPtr) {
129     // ottieni il numero del conto da creare
130     printf("%s", "Enter new account number (1 - 100): ");
131     int account = 0; // numero del conto
132     scanf("%d", &account);
133
134     // sposta il puntatore del file al record corretto nel file
135     fseek(fPtr, (account - 1) * sizeof(struct clientData), SEEK_SET);
136
137     // leggi il record dal file
138     struct clientData client = {0, "", "", 0.0};
139     fread(&client, sizeof(struct clientData), 1, fPtr);
140
141     // stampa un messaggio di errore se il conto esiste già'
142     if (client.account != 0) {
143         printf("Account #%d already contains information.\n",
144               client.account);
145     }
146     else { // crea un record
147         // l'utente inserisce cognome, nome e saldo
148         printf("%s", "Enter lastname, firstname, balance\n? ");
149         scanf("%14s%9s%lf", &client.lastName, &client.firstName,
150               &client.balance);
151
152         client.account = account;
153
154         // sposta il puntatore del file al record corretto nel file
155         fseek(fPtr, (client.account - 1) * sizeof(struct clientData),
156               SEEK_SET);
157
158         // inserisci il record nel file
159         fwrite(&client, sizeof(struct clientData), 1, fPtr);
160     }
161 }
162
163 // cancella un record esistente
164 void deleteRecord(FILE *fPtr) {
```

```

165 // ottieni il numero del conto da cancellare
166 printf("%s", "Enter account number to delete (1 ~ 100): ");
167 int account = 0; // numero del conto
168 scanf("%d", &account);
169
170 // sposta il puntatore del file al record corretto nel file
171 fseek(fPtr, (account - 1) * sizeof(struct clientData), SEEK_SET);
172
173 // leggi il record dal file
174 struct clientData client = {0, "", "", 0.0};
175 fread(&client, sizeof(struct clientData), 1, fPtr);
176
177 // stampa un messaggio di errore se il record non esiste
178 if (client.account == 0) {
179     printf("Account %d does not exist.\n", account);
180 }
181 else { // cancella il record
182     // sposta il puntatore del file al record corretto nel file
183     fseek(fPtr, (account - 1) * sizeof(struct clientData), SEEK_SET);
184
185     struct clientData blankClient = {0, "", "", 0}; // cliente vuoto
186
187     // sostituisci il record esistente con il record vuoto
188     fwrite(&blankClient, sizeof(struct clientData), 1, fPtr);
189 }
190 }
191
192 // consenti all'utente di inserire la scelta del menu
193 int enterChoice(void) {
194     // stampa le opzioni disponibili
195     printf("%s", "\nEnter your choice\n"
196             "1 - store a formatted text file of accounts called\n"
197             "    \"accounts.txt\" for printing\n"
198             "2 - update an account\n"
199             "3 - add a new account\n"
200             "4 - delete an account\n"
201             "5 - end program\n? ");
202
203     int menuChoice = 0; // variabile per memorizzare scelta dell'utente
204     scanf("%d", &menuChoice); // ricevi la scelta dall'utente
205     return menuChoice;
206 }

```

Figura 11.7 Programma per l'elaborazione di transazioni.

Esercizi correlati

Questo caso pratico sul sistema per l'elaborazione di transazioni è supportato dagli Esercizi 11.11 (Inventario di ferramenta) e 11.17 (Sistema per l'elaborazione di transazioni modificato).

✓ Autovalutazione

1. *(Discussione)* Nel seguente codice, che cosa verifica la condizione dell'istruzione if?

```
if ((cfPtr = fopen("accounts.dat", "rb+")) == NULL)
```

Risposta: La condizione verifica se il file accounts.dat è stato aperto con successo in modalità binaria per la lettura e la scrittura.

2. (*Discussione*) Che cosa fa la seguente istruzione nel programma della Figura 11.7?

```
fseek(fPtr, (account - 1) * sizeof(struct clientData), SEEK_SET);
```

Risposta: Questa istruzione sposta il puntatore di posizione per il file che fPtr rappresenta nella posizione per il record clientData account.

11.10 Programmazione sicura in C

fprintf_s e fscanf_s

Gli esempi nei Paragrafi 11.3 e 11.4 usano funzioni fprintf e fscanf, rispettivamente, per scrivere un testo su file e leggere un testo da file. L'Annex K del C standard fornisce versioni di queste funzioni chiamate fprintf_s e fscanf_s che sono identiche alle funzioni printf_s e scanf_s introdotte precedentemente, solo che richiedono anche la specifica di un argomento puntatore FILE che indica il file da manipolare. Se le librerie standard del vostro compilatore C comprendono queste funzioni, vi conviene usarle al posto di fprintf e fscanf. Come con scanf_s e printf_s, le versioni di Microsoft di fprintf_s e fscanf_s differiscono da quelle nell'Annex K.

Capitolo 9 del SEI CERT C Coding Standard

Il Capitolo 9 del *SEI CERT C Coding Standard* è dedicato alle raccomandazioni e alle regole per input/output. Molte riguardano l'elaborazione di file in generale e diverse riguardano le funzioni di elaborazione di file presentate in questo capitolo. Per maggiori informazioni, visitate il sito <https://wiki.sei.cmu.edu/>.

- FIO03-C. Quando si apre un file per scrivere, usando le modalità non esclusive di apertura del file discusse in questo capitolo, se il file esiste la funzione fopen lo apre ed elimina i suoi contenuti, senza fornire indicazioni riguardo all'esistenza del file prima della chiamata di fopen. Per assicurarsi che un file esistente non sarà aperto ed eliminato, si può usare la *modalità esclusiva di scrittura* del C11 (esaminata nel Paragrafo 11.3), che permette a fopen di aprire il file *solo* se non esiste già.
- FIO04-C. Nel codice sviluppato a livello industriale bisogna sempre controllare i valori di ritorno delle funzioni di elaborazione di file che restituiscono indicatori di errore per assicurarsi che le funzioni eseguano i loro compiti correttamente.
- FIO07-C. La funzione rewind non restituisce un valore, pertanto non si può verificare se l'operazione sia riuscita. Si raccomanda, invece, di usare la funzione fseek perché questa, se fallisce, restituisce un valore diverso da zero.
- FIO09-C. In questo capitolo abbiamo illustrato sia i file di testo che i file binari. Per via di differenze nelle rappresentazioni di dati binari su diverse piattaforme, spesso i file scritti in formato binario *non* sono portabili. Per rappresentazioni di file più portabili, prendete in considerazione l'uso di file di testo o di una libreria di funzioni in grado di gestire le differenze nelle rappresentazioni di file binari su diverse piattaforme.
- FIO14-C. Alcune funzioni di libreria non operano in maniera identica sui file di testo e sui file binari. In particolare, *non* si è sicuri che la funzione fseek lavori correttamente con i file binari se si specifica SEEK_END, pertanto è preferibile usare SEEK_SET.
- FIO42-C. Su molte piattaforme è possibile avere solo un numero limitato di file aperti contemporaneamente. Per questa ragione, occorre sempre chiudere un file non appena esso non è più necessario per il proprio programma.

✓ Autovalutazione

1. (*Completere*) Quando aprite un file per la scrittura, potete prevenire che venga eliminato un file esistente usando la _____, che consente a fopen di aprire il file solo se esso non esiste.

Risposta: modalità esclusiva di scrittura.

2. (*Vero/Falso*) La funzione `rewind` non restituisce un valore, pertanto non è possibile verificare che l'operazione abbia avuto successo. Al suo posto, usate la funzione `fseek` poiché restituisce un valore diverso da zero se fallisce.

Risposta: *Vero*.

3. (*Vero/Falso*) Su molte piattaforme è possibile avere solo un numero limitato di file aperti contemporaneamente. Per questa ragione, occorre sempre chiudere un file non appena esso non è più necessario.

Risposta: *Vero*.

11.11 Riepilogo

Paragrafo 11.1 Introduzione

- I file sono usati per la memorizzazione permanente di grandi quantità di dati.
- I computer memorizzano i file su **dispositivi di memoria secondaria**, come unità allo stato solido, unità flash e dischi rigidi.

Paragrafo 11.2 File e stream

- Il C vede ciascun file come uno **stream di byte** sequenziale. Quando un file viene aperto, a esso viene associato uno stream.
- Tre stream vengono aperti automaticamente quando inizia l'esecuzione di un programma: lo **standard input**, lo **standard output** e lo **standard error**.
- Gli stream forniscono **canali di comunicazione** tra file e programmi.
- Lo **stream standard input** permette a un programma di **leggere dati dalla tastiera**, e lo **stream standard output** permette a un programma di **stampare dati sullo schermo**.
- L'apertura di un file restituisce un puntatore a una **struttura FILE** (definita in `<stdio.h>`) contenente informazioni utilizzate per elaborare il file. Questa struttura include un **descrittore di file**, cioè un indice per un array del sistema operativo chiamato **tavella dei file aperti**. Ogni elemento dell'array contiene un **blocco di controllo del file (FCB)** che il sistema operativo usa per amministrare un particolare file.
- Lo standard input, lo standard output e lo standard error vengono manipolati usando i puntatori a file predefiniti `stdin`, `stdout` e `stderr`.
- La funzione `fgetc` **legge un carattere** da un file. Essa riceve come argomento un puntatore FILE per il file da cui verrà letto un carattere.
- La funzione `fputc` **scrive un carattere** su un file. Essa riceve come argomenti un carattere da scrivere e un puntatore FILE per il file su cui verrà scritto il carattere.
- Le funzioni `fgets` e `fputs`, rispettivamente, **leggono una riga** da un file e **scrivono una riga** su un file.

Paragrafo 11.3 Creazione di un file ad accesso sequenziale

- Il C non assegna alcuna struttura a un file. È necessario specificare esplicitamente una struttura per un file per soddisfare le esigenze di una particolare applicazione.
- Un programma in C amministra ogni file con una struttura di tipo FILE separata.
- Ogni file aperto deve avere un **puntatore FILE** dichiarato separatamente che viene usato come riferimento al file.
- La funzione `fopen` riceve come argomenti un nome di file e una **modalità di apertura del file** e restituisce un puntatore alla struttura FILE per il file aperto o NULL se non è stato possibile aprire il file.
- La **modalità di apertura del file "w"** indica che il file deve essere aperto per la scrittura. Se il file non esiste, `fopen` lo crea. Se il file esiste, il contenuto viene eliminato senza avvertimento.

- La funzione `feof` riceve un puntatore a un `FILE` e restituisce un valore diverso da zero (vero) quando trova impostato l'indicatore di end-of-file; diversamente, la funzione restituisce zero. Ogni tentativo di lettura da un file per il quale `f.eof` restituisce vero fallirà.
- La funzione `fprintf` è equivalente a `printf`, ma inoltre riceve anche come argomento un puntatore a file per il file su cui saranno scritti i dati.
- La funzione `fclose` riceve come argomento un puntatore a file e chiude il file specificato.
- Quando viene aperto un file, il blocco di controllo del file (FCB) per il file viene copiato nella memoria. Il FCB è usato dal sistema operativo per amministrare il file.
- Per leggere un file esistente, apritelo per la lettura ("r").
- Per aggiungere record alla fine di un file esistente, aprite il file in modalità *append* ("a").
- Per aprire un file così da poterci scrivere e leggere, usate una **modalità di aggiornamento**: "r+", "w+" o "a+". La modalità "r+" apre un file per la lettura e la scrittura. La modalità "w+" crea un file per la lettura e la scrittura, ma il contenuto di un file esistente viene eliminato. La modalità "a+" apre un file per la lettura e la scrittura; tutta la scrittura è effettuata alla fine del file. Se il file non esiste, viene creato.
- Ogni modalità di apertura di un file ha una **modalità binaria** (b) corrispondente per manipolare i **file binari**.
- La **modalità esclusiva di scrittura** assicura che un file esistente non venga sovrascritto. Se l'apertura di un file nella modalità esclusiva di scrittura riesce e il sistema sottostante supporta l'accesso esclusivo al file, allora solo il vostro programma può accedere al file finché è aperto.

Paragrafo 11.4 Lettura di dati da un file ad accesso sequenziale

- La funzione `fscanf` è equivalente alla funzione `scanf`, ma riceve come argomento un puntatore a file per il file da cui si leggono i dati.
- La funzione `rewind` risposiziona il **puntatore di posizione del file** di un programma all'inizio del file (cioè al byte 0) puntato dal suo argomento.
- Il puntatore di posizione del file è un valore intero che specifica la posizione espressa in byte nel file oggetto della successiva lettura o scrittura. Questo è a volte detto **offset del file**. Il puntatore di posizione del file è un membro della struttura `FILE` associata a ogni file.
- I dati di un file sequenziale non possono essere modificati senza il rischio di distruggere altri dati del file.

Paragrafo 11.5 File ad accesso casuale

- I **file ad accesso casuale** usano record di lunghezza fissa accessibili direttamente senza dover effettuare ricerche in altri record.
- Poiché ogni record in un file ad accesso casuale ha normalmente la stessa lunghezza, la posizione esatta di un record rispetto all'inizio del file può essere calcolata come una funzione della **chiave del record**.
- I record di lunghezza fissa permettono di inserire dati in un file ad accesso casuale senza distruggere altri dati. I dati memorizzati in precedenza possono anche essere aggiornati o cancellati senza riscrivere l'intero file.

Paragrafo 11.6 Creazione di un file ad accesso casuale

- La funzione `fwrite` trasferisce un numero specificato di byte su un file iniziando da una posizione specificata in memoria. I dati vengono scritti a iniziare dal punto nel file indicato dal puntatore di posizione del file.
- La funzione `fread` trasferisce un numero specificato di byte dalla posizione nel file indicata dal puntatore di posizione del file a un'area nella memoria a cominciare da un indirizzo specificato.
- Le funzioni `fwrite` e `fread` possono **leggere e scrivere array di dati** da e su file. Il terzo argomento sia di `fread` che di `fwrite` è il numero di elementi da elaborare.
- I programmi di elaborazione di file normalmente scrivono un elemento `struct` alla volta.

Paragrafo 11.7 Scrittura di dati in maniera casuale su un file ad accesso casuale

- La funzione `fseek` riposiziona il puntatore di posizione di un dato file a una posizione specifica in termini di byte. Il suo secondo argomento indica il numero di byte di cui spostarsi e il suo terzo argomento indica la posizione di riferimento. Il terzo argomento può essere `SEEK_SET`, `SEEK_CUR` o `SEEK_END`. `SEEK_SET` indica che il riferimento per l'accesso ai dati è l'inizio del file, `SEEK_CUR` indica che è la posizione corrente nel file e `SEEK_END` indica che è la fine del file.
- I programmi realizzati a livello industriale devono determinare se funzioni come `fscanf`, `fseek` e `fwrite` operano correttamente controllando i loro valori di ritorno.
- La funzione `fscanf` restituisce il numero di campi letti con successo o il valore `EOF` se si verifica un problema durante la lettura dei dati.
- La funzione `fseek` restituisce un valore diverso da zero se l'operazione non può essere eseguita.
- La funzione `fwrite` restituisce il numero di elementi che ha inviato in uscita con successo. Se questo numero è minore del terzo argomento nella chiamata della funzione, vuol dire che si è verificato un errore.

Paragrafo 11.8 Lettura di dati da un file ad accesso casuale

- La funzione `fread` legge un numero specificato di byte da un file nella memoria.
- La funzione `fread` può leggere diversi elementi di un array di lunghezza fissa fornendo un puntatore all'array nel quale saranno memorizzati gli elementi e indicando il numero di elementi da leggere.
- La funzione `fread` restituisce il numero di elementi che ha letto con successo. Se questo numero è minore del terzo argomento nella chiamata della funzione, si è verificato un errore di lettura.

Esercizi di autovalutazione

11.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- La funzione _____ chiude un file.
- La funzione _____ legge i dati da un file in un modo simile a come `scanf` legge da `stdin`.
- La funzione _____ legge un carattere da un file specificato.
- La funzione _____ legge una riga da un file specificato.
- La funzione _____ apre un file.
- La funzione _____ è usata normalmente quando si leggono dati da un file in applicazioni che richiedono l'accesso casuale.
- La funzione _____ ricolloca il puntatore di posizione del file in un punto specifico del file.

11.2 Stabilite quali delle seguenti affermazioni sono *vere* e quali *false*. Se *false*, spiegate perché.

- La funzione `fscanf` non può essere usata per leggere dati dallo standard input.
- Si deve esplicitamente usare `fopen` per aprire gli stream di standard input, standard output e standard error.
- Un programma deve chiamare esplicitamente la funzione `fclose` per chiudere un file.
- Se il puntatore di posizione del file punta a una posizione in un file sequenziale diversa dall'inizio del file, il file deve essere chiuso e riaperto per leggerlo dall'inizio.
- La funzione `fprintf` può scrivere sullo standard output.
- I dati nei file ad accesso sequenziale vengono sempre aggiornati senza sovrascrivere altri dati.
- Non è necessario esaminare tutti i record in un file ad accesso casuale per trovare un record specifico.
- I record nei file ad accesso casuale non sono di lunghezza uniforme.
- La funzione `fseek` può solo avere come riferimento l'inizio di un file.

11.3 Scrivete una singola istruzione per effettuare ognuna delle seguenti operazioni. Supponete che ciascuna di queste istruzioni si riferisca allo stesso programma.

- Aprite il file "oldmast.dat" per la lettura e assegnate il puntatore a file restituito a `ofPtr`.
- Aprite il file "trans.dat" per la lettura e assegnate il puntatore a file restituito a `tfPtr`.

- c) Aprite il file "newmast.dat" per la scrittura (e la creazione) e assegnate il puntatore a file restituito a nFptr.
- d) Leggete un record dal file "oldmast.dat". Il record è costituito dall'intero account, dalla stringa name e dal valore in virgola mobile currentBalance.
- e) Leggete un record dal file "trans.dat". Il record è costituito dall'intero account e dal valore in virgola mobile dollarAmount.
- f) Scrivete un record sul file "newmast.dat". Il record è costituito dall'intero account, dalla stringa name e da valore in virgola mobile currentBalance.
- 11.4 Trovate l'errore in ognuno dei seguenti segmenti di programma e spiegate come correggerlo.
- Il file a cui fPtr fa riferimento ("payables.dat") non è stato aperto.
- ```
printf(fPtr, "%d%s%d\n", account, company, amount);
```
- open("receive.dat", "r+");
  - L'istruzione seguente deve leggere un record dal file "payables.dat". Il puntatore a file payPtr fa riferimento a questo file e il puntatore a file recPtr fa riferimento al file "receive.dat":
- ```
scanf(recPtr, "%d%s%d\n", &account, company, &amount);
```
- Il file "tools.dat" deve essere aperto per aggiungervi dati senza eliminare quelli correnti.
- ```
if ((tfPtr = fopen("tools.dat", "w")) != NULL)
```
- Il file "courses.dat" deve essere aperto in modalità *append* senza modificarne i contenuti correnti.
- ```
if ((cfPtr = fopen("courses.dat", "w+")) != NULL)
```

Risposte agli esercizi di autovalutazione

- 11.1 a) fclose. b) fscanf. c) fgetc. d) fgets. e) fopen. f) fread. g) fseek.
- 11.2 a) Falso. La funzione fscanf può essere usata per leggere dallo standard input includendo il puntatore allo stream standard input, stdin, nella sua chiamata.
- b) Falso. Questi tre stream sono aperti automaticamente dal C quando inizia l'esecuzione del programma.
- c) Falso. I file verranno chiusi quando l'esecuzione del programma terminerà, ma tutti i file vanno esplicitamente chiusi con fclose.
- d) Falso. È possibile usare la funzione rewind per ricollocare il puntatore di posizione del file all'inizio del file.
- e) Vero.
- f) Falso. Nella più parte dei casi, i record di file sequenziali non sono di lunghezza uniforme. Pertanto, è possibile che l'aggiornamento di un record porti a sovrascrivere altri dati.
- g) Vero.
- h) Falso. I record in un file ad accesso casuale sono normalmente di lunghezza uniforme.
- i) Falso. È possibile avere come riferimento l'inizio del file, la fine del file o la posizione corrente nel file.
- 11.3 a) ofPtr = fopen("oldmast.dat", "r");
 b) tfPtr = fopen("trans.dat", "r");
 c) nfPtr = fopen("newmast.dat", "w");
 d) fscanf(ofPtr, "%d%s%f", &account, name, ¤tBalance);
 e) fscanf(tfPtr, "%d%f", &account, &dollarAmount);
 f) fprintf(nfPtr, "%d %s %.2f", account, name, currentBalance);
- 11.4 a) Errore: il file "payables.dat" non è stato aperto prima dell'utilizzo di fPtr.
 Correzione: usate fopen per aprire "payables.dat" per la scrittura, l'aggiunta di dati in coda e l'aggiornamento.

- b) Errore: la funzione open non è una funzione del C standard.
Correzione: usate la funzione fopen.
- c) Errore: la funzione scanf deve essere fscanf. La funzione fscanf usa il puntatore a file sbagliato per fare riferimento al file "payables.dat".
Correzione: usate payPtr per fare riferimento a "payables.dat" e usate fscanf.
- d) Errore: il contenuto del file viene eliminato perché il file viene aperto in scrittura ("w").
Correzione: per aggiungere dati al file, aprite il file o per l'aggiornamento ("r+") o in modalità append ("a" o "a+").
- e) Errore: il file "courses.dat" viene aperto per l'aggiornamento in modalità "w+", che elimina il contenuto corrente del file.
Correzione: aprite il file in modalità "a" o "a+".

Esercizi

11.5 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- a) I computer memorizzano grandi quantità di dati su dispositivi secondari di memoria come _____.
 b) Un _____ è composto da diversi campi.
 c) Per facilitare il recupero di record specifici da un file, un campo in ogni record viene scelto come _____.
 d) Un gruppo di caratteri collegati che trasmette un significato è chiamato _____.
 e) I puntatori a file per i tre stream che vengono aperti automaticamente quando inizia l'esecuzione di un programma sono denominati _____, _____ e _____.
 f) La funzione _____ scrive un carattere su un file specificato.
 g) La funzione _____ scrive una riga su un file specificato.
 h) La funzione _____ è usata generalmente per scrivere dati su un file ad accesso casuale.
 i) La funzione _____ ricolloca il puntatore di posizione del file all'inizio del file.

11.6 (Creazione di dati per un programma di confronto di file) Scrivete un semplice programma per creare alcuni dati di test per verificare il programma dell'Esercizio 11.7. Usate il seguente campione di dati relativi a conti di clienti.

File principale			File delle transazioni	
Numero di conto	Nome	Saldo	Numero di conto	Ammontare in dollari
100	Alan Jones	348.17	100	27.14
300	Mary Smith	27.19	300	62.11
500	Sam Sharp	0.00	400	100.56
700	Suzy Green	-14.22	900	82.17

11.7 (Confronto di file) L'Esercizio 11.3 chiedeva al lettore di scrivere una serie di istruzioni singole. In realtà, queste istruzioni formano il nucleo di un importante tipo di programma di elaborazione di file, vale a dire un programma di confronto di file. Nell'elaborazione di dati commerciali è comune avere sistemi con diversi file. In un sistema di contabilità dei clienti, per esempio, vi è generalmente un file principale contenente informazioni dettagliate su ogni cliente, come il nome del cliente, l'indirizzo, il numero di telefono, il saldo scoperto, il limite di credito, i termini dello sconto, gli accordi di contratto ed eventualmente una sintesi dei recenti acquisti e pagamenti in contanti.

Quando avvengono delle transazioni (cioè si effettuano vendite e si ricevono pagamenti in contanti), le informazioni che le riguardano sono inserite in un file. Alla fine di ogni periodo di attività (cioè un mese per alcune aziende, una settimana per altre e un giorno in alcuni casi), il file delle transazioni (chiamato "trans.dat" nell'Esercizio 11.3) viene confrontato con il file principale (chiamato "oldmast.dat" nell'Esercizio 11.3), per aggiornare il record acquisti e pagamenti di ciascun conto. Durante l'aggiornamento, il file principale viene

riscritto come nuovo file ("newmast.dat"), che viene poi usato alla fine del successivo periodo di attività per iniziare nuovamente il processo di aggiornamento.

I programmi di confronto di file incorrono in alcuni problemi che non si manifestano nei programmi che usano un singolo file. Per esempio, non sempre si verifica una corrispondenza tra elementi dei file. Se un cliente nel file principale non ha fatto alcun acquisto o pagamento nel periodo corrente, nel file delle transazioni non comparirà alcun record che lo riguarda. Analogamente, un cliente che ha fatto alcuni acquisti o pagamenti in contanti potrebbe essere venuto a far parte di quella comunità di recente e l'azienda può non aver avuto la possibilità di creare un record principale per questo cliente.

Usate le istruzioni dell'Esercizio 11.3 come base per un programma completo di contabilità dei clienti, in grado di effettuare confronti tra file. Usate il numero di conto per ognuno dei file come chiave del record per i confronti. Supponete che ogni file sia un file sequenziale con record memorizzati in ordine crescente di numero di conto.

Quando un confronto ha successo (cioè i record con lo stesso numero di conto compaiono sia sul file principale che sul file delle transazioni), aggiungete l'ammontare in dollari contenuto nel file delle transazioni al saldo corrente nel file principale e scrivete il record nel file "newmast.dat". (Supponete che gli acquisti siano indicati con importi positivi nel file delle transazioni e che i pagamenti siano indicati con importi negativi.) Quando è presente un record principale per un certo conto ma non vi è alcun record di transazione corrispondente, trasferite solamente il record principale su "newmast.dat". Quando è presente un record di transazione ma non vi è alcun record principale corrispondente, stampate il messaggio "Unmatched transaction record for account number ..." (inserire il numero di conto dal record della transazione).

11.8 (Verificare gli esercizi per il confronto di file) Eseguite il programma dell'Esercizio 11.7 usando i file di dati di test creati nell'Esercizio 11.6. Controllate attentamente i risultati.

11.9 (Confronto di file con transazioni multiple) È possibile (in realtà comune) avere diversi record di transazioni con la stessa chiave. Questo succede perché, durante un periodo di attività commerciale, un particolare cliente potrebbe fare diversi acquisti e pagamenti in contanti. Riscrivete il vostro programma di confronto di file per la contabilità dei clienti dell'Esercizio 11.7 per rendere possibile il trattamento di record relativi a transazioni diverse ma con la stessa chiave. Modificate i dati di test dell'Esercizio 11.6 per includere i seguenti ulteriori record di transazione:

Numero di conto	Ammontare in dollari
300	83.89
700	80.78
700	1.53

11.10 (Scrivere istruzioni per eseguire un'operazione) Scrivete delle istruzioni che eseguano ognuna delle seguenti operazioni. Supponete che sia stata definita la struttura

```
struct person {
    char lastName[15];
    char firstName[15];
    char age[4];
};
```

e che il file sia già aperto per la scrittura.

- Inizializzate il file "nameage.dat" affinché vi siano 100 record con lastName = "unassigned", firstname = "" e age = "0".
- Inserite 10 cognomi, nomi ed età e scriveteli sul file.
- Aggiornate un record; se nel record non vi sono informazioni, dite all'utente "No info".
- Cancellate un record con informazioni reinizializzando quel particolare record.

11.11 (Inventario di ferramenta) Siete i proprietari di un negozio di ferramenta e avete necessità di tenere un inventario che possa dirvi quali attrezzi avete, quanti ne avete e il costo di ognuno di essi. Scrivete un programma che inizializzi il file "hardware.dat" con 100 record vuoti, vi faccia inserire i dati riguardanti ogni attrezzo, vi consenta di fare una lista di tutti i vostri attrezzi, vi faccia cancellare un record relativo a un attrezzo che non avete più e vi faccia aggiornare una *qualsiasi* informazione nel file. Il numero di identificazione dell'attrezzo deve essere il numero del record. Usate le seguenti informazioni per riempire il file:

Record #	Nome attrezzo	Quantità	Costo
3	Levigatrice elettrica	7	57.98
17	Martello	76	11.99
24	Seghetto	21	11.00
39	Tagliaerba	3	79.50
56	Sega elettrica	18	99.99
68	Cacciavite	106	6.99
77	Martello da fabbro	11	21.50
83	Chiave inglese	34	7.50

11.12 (Generatore di parole per numeri telefonici) Le tastiere telefoniche standard contengono le cifre da 0 a 9. A tutti i numeri da 2 a 9 sono associate tre lettere, come indica la seguente tabella:

Cifra	Lettera	Cifra	Lettera
2	A B C	6	M N O
3	D E F	7	P R S
4	G H I	8	T U V
5	J K L	9	W X Y

Molte persone trovano difficile memorizzare i numeri di telefono, e così usano la corrispondenza tra cifre e lettere per generare parole di sette lettere corrispondenti ai loro numeri di telefono. Per esempio, una persona il cui numero telefonico è 686-2377 potrebbe usare la corrispondenza indicata nella tabella riportata sopra per generare la parola di sette lettere "NUMBERS".

Le aziende cercano frequentemente di avere numeri telefonici facili da ricordare per i loro clienti. Se un'azienda può rendere nota ai suoi clienti una parola semplice da comporre, senza dubbio riceverà qualche telefonata in più.

Ogni parola di sette lettere corrisponde esattamente a un numero telefonico di sette cifre. Il ristorante che desidera accrescere i propri affari sui piatti da asporto, potrebbe sicuramente farlo con il numero 825-3688 (cioè, "TAKEOUT").

Ogni numero di telefono di sette cifre corrisponde a molte parole diverse di sette lettere. Purtroppo, la maggior parte di queste rappresenta giustapposizioni irriconoscibili di lettere. È possibile, tuttavia, che al proprietario di un salone da barba faccia piacere sapere che il numero telefonico del proprio negozio 424-7288 corrisponde a "HAIRCUT". Il proprietario di un negozio di alcolici si rallegrerebbe senza dubbio se il numero telefonico del proprio negozio 233-7226 corrispondesse a "BEERCAN". Un veterinario con il numero di telefono 738-2273 sarebbe contento di sapere che il numero corrisponde alle lettere "PETCARE".

Scrivete un programma in C che, dato un numero di sette cifre, scriva su un file ogni possibile parola di sette lettere corrispondente a quel numero. Ne esistono 2187 (3 alla settima potenza) di tali parole. Evitate numeri di telefono con le cifre 0 e 1.

11.13 (Progetto: modifiche al generatore di parole per numeri telefonici) Se avete a disposizione un dizionario computerizzato, modificate il programma che avete scritto nell'Esercizio 11.12 per prendere le parole dal dizionario. Alcune combinazioni di sette lettere create da questo programma corrispondono a due o più parole (per esempio, il numero telefonico 843-2677 produce "THEBOSS").

11.14 (Usare funzioni per l'elaborazione di file con stream di input/output standard) Modificate l'esempio della Figura 8.8 in modo da usare le funzioni `fgetc` e `fputs` al posto di `getchar` e `puts`. Il programma deve offrire all'utente l'opzione di poter leggere dallo standard input e scrivere sullo standard output, oppure di leggere da un file specificato e scrivere su un file specificato. Se l'utente sceglie la seconda opzione, fate sì che inserisca i nomi dei file di input e di output.

11.15 (Inviare in uscita su un file le dimensioni dei tipi) Scrivete un programma che usi l'operatore `sizeof` per determinare le dimensioni in byte dei vari tipi di dati sul vostro computer. Scrivete i risultati sul file "datasize.dat" così che dopo possiate stampare i risultati. Il formato per i risultati nel file deve essere il seguente (le dimensioni dei tipi sul vostro computer potrebbero essere diverse da quelle mostrate nell'output dell'esempio):

Data type	Size
<code>char</code>	1
<code>unsigned char</code>	1
<code>short int</code>	2
<code>unsigned short int</code>	2
<code>int</code>	4
<code>unsigned int</code>	4
<code>long int</code>	4
<code>unsigned long int</code>	4
<code>float</code>	4
<code>double</code>	8
<code>long double</code>	16

11.16 (Simpletron con elaborazione di file) Nell'Esercizio 7.29, avete scritto un programma per la simulazione software di un computer che usava uno speciale linguaggio macchina chiamato Simpletron Machine Language (SML). Nella simulazione, ogni volta che volevate avviare l'esecuzione di un programma in SML dovevate inserire il programma nel simulatore dalla tastiera. Se scrivendo il programma in SML commettevate un errore, era necessario far ripartire il simulatore per reinserire il codice SML. Sarebbe bello poter leggere da un file il programma in SML, invece di scriverlo ogni volta, in modo da risparmiare tempo e ridurre gli errori nella preparazione dell'esecuzione di programmi in SML.

- Modificate il simulatore che avete scritto nell'Esercizio 7.29 per leggere programmi in SML da un file specificato dall'utente alla tastiera.
- Dopo aver terminato l'esecuzione, il Simpletron invia in uscita i contenuti dei suoi registri e della sua memoria sullo schermo. Non sarebbe male memorizzare l'output in un file. Pertanto modificate il simulatore per memorizzare il suo output in un file, oltre a stamparlo sullo schermo.

11.17 (Sistema per l'elaborazione di transazioni modificato) Modificate il programma del Paragrafo 11.9 per includere un'opzione che stampi l'elenco di account sullo schermo. Considerate la modifica della funzione `textFile` per usare l'output standard o un file di testo basato su un parametro della funzione addizionale che specifichi dove deve essere scritto l'output.

11.18 (Progetto: phishing scanner) Il *phishing* è una forma di furto di identità, in cui in un'e-mail un mittente, spacciandosi per una fonte fidata, tenta di acquisire informazioni private, come il nome utente, la password, i numeri della carta di credito e il numero del codice fiscale. Le e-mail di raggiro telematico che sostengono di provenire da note banche, da società di carte di credito, siti di vendite all'asta, social network e servizi di pagamento online possono sembrare del tutto legittime. Questi messaggi fraudolenti forniscono spesso link a siti web fasulli dove vi si chiede di inserire informazioni sensibili.

Visitate <https://snopes.com> e altri siti web per trovare le liste delle principali frodi telematiche. Controllate anche l'Anti-Phishing Working Group (<https://apwg.org/>) e il sito del Cyber Investigations dell'FBI (<https://www.fbi.gov/investigate/cyber>), dove troverete informazioni sulle truffe più recenti e su come proteggervi.

Create una lista di 30 parole, frasi e nomi di aziende che si trovano comunemente nei messaggi di phishing. Assegnate un punteggio a ognuno di essi valutando voi stessi il livello di probabilità che facciano parte di un messaggio di phishing (per esempio: un punto se è leggermente probabile, due punti se è moderatamente probabile o tre punti se è altamente probabile). Scrivete un programma che esegua la scansione di un file di testo per questi termini e frasi. Ogni volta che una parola chiave o una frase ricorre nel file di testo, aggiungete il punteggio assegnato ai punti totali per quella parola o frase. Per ogni parola chiave o frase trovata, stampate una riga con la parola o la frase, il numero delle volte che ricorre e il punteggio totale. Poi mostrate il punteggio totale per l'intero messaggio. Il vostro programma assegna un punteggio totale alto ad alcune vere e-mail di phishing che avete ricevuto? Assegna un punteggio totale alto ad alcune e-mail legittime che avete ricevuto?

Caso pratico sull'intelligenza artificiale: introduzione alla NLP – Chi ha scritto le opere di Shakespeare?

11.19 (Introduzione all'elaborazione del linguaggio naturale e al rilevamento della similarità) Ogni giorno usiamo il linguaggio naturale in diverse forme di comunicazione, tra cui, per esempio:

- leggere i messaggi di testo e controllare le ultime notizie;
- parlare a familiari, amici e colleghi e ascoltare le loro risposte;
- comunicare tramite il linguaggio dei segni con un amico sordo, che ama guardare programmi video sottotitolati;
- in caso di disabilità visiva, leggere in braille, ascoltare gli audiolibri e usare uno screen reader che legge dallo schermo del computer;
- leggere le e-mail, distinguendo tra posta indesiderata e comunicazioni importanti;
- ricevere da un cliente un'e-mail in spagnolo e utilizzare un programma gratuito per tradurla, e rispondere poi in italiano, sapendo che il cliente può facilmente ritradurla a sua volta in spagnolo;
- guidare, osservando i segnali stradali quali "Stop," "Limite di velocità 50" e "Lavori in corso";
- dare comandi verbali alla propria auto, come "chiama casa" o "suona musica classica", oppure porre domande come "dov'è la stazione di servizio più vicina?";
- insegnare a un bambino a parlare e a leggere;
- apprendere una lingua straniera.

L'elaborazione del linguaggio naturale (NLP, *Natural Language Processing*) aiuta i computer a comprendere, analizzare ed elaborare testi e discorsi umani. L'elaborazione del linguaggio naturale viene eseguita su raccolte di testi composte da tweet, post di Facebook, conversazioni, recensioni di film, opere di Shakespeare, documenti storici, notizie, verbali di riunioni, e molto altro ancora. Una raccolta di testi è detta *corpus*, al plurale *corpora*.

Elenchiamo qui di seguito alcune applicazioni fondamentali della NLP.

- **Comprensione del linguaggio naturale:** comprende il contenuto di testi o del linguaggio parlato.
- **Analisi del sentimento:** determina se un testo ha un tono positivo, neutrale o negativo. Per esempio, le aziende analizzano il sentimento dei tweet relativi ai loro prodotti.
- **Valutazione della leggibilità:** determina il grado di leggibilità di un testo, in base a vocabolario utilizzato, lunghezza delle parole, lunghezza e struttura delle frasi, argomenti trattati ecc. Nello scrivere questo libro abbiamo usato la versione a pagamento dello strumento Grammarly² (NPL) come aiuto per mettere a punto la stesura del testo e assicurarci che fosse leggibile per una vasta platea di lettori.

2. Esiste anche una versione gratuita di Grammarly (<https://www.grammarly.com>).

- **Assistenti virtuali intelligenti:** software che assiste nell'esecuzione delle attività quotidiane. Tra gli assistenti virtuali intelligenti più popolari vi sono Alexa di Amazon, Siri di Apple, Cortana di Microsoft e l'assistente di Google.
- **Riassunto di testi:** riassume i punti fondamentali di un testo esteso. Permette un notevole risparmio di tempo alle persone molto impegnate.
- **Riconoscimento vocale:** converte in testo il linguaggio parlato.
- **Sintesi vocale:** converte in linguaggio parlato un testo.
- **Identificazione della lingua:** riceve un testo in una lingua non nota a priori e identifica automaticamente la lingua in cui è scritto.
- **Traduzione:** converte il testo in altre lingue.
- **Riconoscimento di entità nominate:** identifica e classifica elementi come date, espressioni temporali, quantità, posti, persone, cose, organizzazioni ecc.
- **Chatbot:** software basato su intelligenza artificiale, con cui le persone interagiscono tramite il linguaggio naturale. L'assistenza clienti automatizzata è un'applicazione molto diffusa dei chatbot.
- **Rilevamento della similarità:** esamina i documenti per determinare quanto sono simili tra loro. Le metriche basilari di similarità includono la lunghezza media delle frasi e la loro distribuzione di frequenza, la lunghezza media delle parole e la loro distribuzione di frequenza, la distribuzione di frequenza dell'uso delle parole, ecc.

Le applicazioni di cui sopra sono supportate nello svolgimento delle loro attività da funzionalità NLP di livello inferiore, tra cui le seguenti.

- **Tokenizzazione:** suddivide il testo in **token**, cioè elementi significativi, quali parole e numeri.
- **POS tagging (*Parts-of-speech tagging*):** identifica la categoria grammaticale di ogni parola, come nome, verbo, aggettivo ecc.
- **Estrazione di frasi nominali:** individua gruppi di parole che rappresentano nomi, come “red brick factory”.³
- **Controllo ortografico e correzione ortografica.**
- **Stemming:** riduce le parole alla loro radice, rimuovendo prefissi o suffissi; per esempio la radice di “corriamo” è “corr” (in inglese, la radice di “varieties” è “variety”).
- **Lematizzazione:** processo analogo allo stemming, ma produce parole reali basandosi sul contesto della parola originale; per esempio il lemma di “corriamo” è “correre” (in inglese, il lemma di “varieties” è “variety”).
- **Conteggio della frequenza delle parole:** determina quante volte ciascuna parola appare in un *corpus*.
- **Eliminazione delle “stop-word”:** rimuove le parole poco significative, come articoli, preposizioni e congiunzioni, per concentrare l'analisi sulle parole più rilevanti in un *corpus*.
- **N-grammi:** produzione di sequenze di parole consecutive in un *corpus*, per identificare le parole che appaiono frequentemente una accanto all'altra; gli n-grammi sono comunemente usati per la previsione della parola successiva negli input di testo, come accade con le parole suggerite dallo smartphone quando si digita un messaggio.

L'esercizio di questo caso pratico ha due scopi:

- il primo è introdurre l'elaborazione del linguaggio naturale, soggetto di importanza cruciale nel più vasto tema dell'intelligenza artificiale, che avrà un ruolo fondamentale nel futuro di chiunque stia imparando oggi la programmazione;

3. La frase “red brick factory” aiuta a capire come mai il linguaggio naturale è un soggetto complicato. Una “red brick factory” è una fabbrica che produce mattoni rossi? È una fabbrica rossa che produce mattoni di qualsiasi colore? È una fabbrica costruita con mattoni rossi che realizza prodotti di qualsiasi tipo? Nel mondo di oggi, potrebbe anche essere il nome di una rock band oppure il nome di un gioco per smartphone.

- il secondo introduce un sottoargomento della NLP, il rilevamento della similarità, che eseguirete utilizzando semplici tecniche di elaborazione di array, stringhe e file.

Progetto Gutenberg

Un'ottima fonte di testi da analizzare è l'enorme collezione di e-book gratuiti del **Progetto Gutenberg**:

<https://www.gutenberg.org>

Il sito contiene più di 60.000 e-book in vari formati, inclusi file di testo puro. Questi libri non sono più protetti da copyright negli Stati Uniti. Per informazioni sui termini d'uso del Progetto Gutenberg e sul copyright in altri Paesi, consultate:

https://www.gutenberg.org/policy/terms_of_use.html

Per questo esercizio, userete i file di testo dell'e-book *Romeo and Juliet* di William Shakespeare:

<https://www.gutenberg.org/ebooks/1513>

e dell'e-book *Edward the Second* di Christopher Marlowe:

<https://www.gutenberg.org/ebooks/20288>

Entrambi sono scaricabili gratuitamente dal sito del Progetto Gutenberg.

Scaricare e-book dal Progetto Gutenberg

Non è consentito l'accesso programmatico agli e-book del Progetto Gutenberg. Dovete scaricare i libri sul vostro sistema prima di analizzarli.⁴ Per scaricare *Romeo and Juliet* come file di testo (*plain-text*), fate clic con il tasto destro del mouse sul link **Plain Text UTF-8** che troverete nella pagina web dell'opera, quindi selezionate

- **Save Link As...** (Chrome/Firefox/Microsoft Edge) o
- **Download Linked File As...** (Safari)

per salvare l'opera nella stessa cartella in cui salverete la soluzione di questo esercizio. Salvate i file con i nomi *RomeoAndJuliet.txt* e *EdwardTheSecond.txt*.

Chi ha scritto le opere di William Shakespeare?

Alcuni studiosi ritengono che le opere di William Shakespeare siano in realtà state scritte da Christopher Marlowe, Sir Francis Bacon o da altri. Potete trovare maggiori informazioni su questo dibattito sul sito:

https://en.wikipedia.org/wiki/Shakespeare_authorship_question

Potete iniziare a confrontare le opere di Shakespeare con quelle di altri autori, utilizzando alcune semplici tecniche di rilevamento della similarità. In questo esercizio, l'obiettivo finale consiste nell'eseguire il rilevamento della similarità tra *Romeo and Juliet* e *Edward the Second* per determinare se Christopher Marlowe potrebbe essere l'autore delle opere di Shakespeare. Se vi appassionerete alla questione, potrete approfondire usando tecniche di rilevamento della similarità più sofisticate.

Analisi di *Romeo and Juliet* per poter eseguire un semplice rilevamento della similarità

Eseguirete ora alcune semplici analisi statistiche in preparazione al processo di determinazione della similarità tra documenti. Inizierete concentrandovi su *Romeo and Juliet* di Shakespeare. In seguito, eseguirete le stesse operazioni con il testo di *Edward the Second*, per poi confrontare i risultati delle vostre analisi. Vi suggeriamo, come controllo, di analizzare anche l'opera di un terzo autore, non coinvolto nella controversia. Quando leggete ed elaborate *Romeo and Juliet*, identificate gli elementi elencati, che poi userete per mostrare le varie statistiche:

- numero totale delle frasi;
- numero totale delle parole;

4. "Information About Robot Access to our Pages," Accesso 1 gennaio 2021. https://www.gutenberg.org/policy/robot_access.html.

- numero totale dei caratteri;
- numero delle frasi di ciascuna lunghezza;
- numero delle parole di ciascuna lunghezza;
- frequenze delle singole parole.

"Pulire" Romeo and Juliet prima dell'analisi

Non sempre i dati sono già pronti per l'analisi. Per esempio, potrebbero avere un formato sbagliato. I data scientist impiegano una gran parte del loro tempo nella preparazione dei dati prima di eseguirne l'analisi. Il processo di preparazione dei dati prima dell'analisi è chiamato *data munging* o *data wrangling*.

Ogni e-book scaricato dal Progetto Gutenberg contiene informazioni e paragrafi di contenuto legale che non devono essere inclusi nelle vostre analisi. Dovete quindi aprire *Romeo and Juliet* in un editor di testi e "pulirlo" rimuovendo il testo del Progetto Gutenberg. In particolare, eliminate tutta la parte dall'inizio del documento fino al titolo, compreso, "THE TRAGEDY OF ROMEO AND JULIET", poi eliminate tutto a partire dal testo che segue fino alla fine del file:

```
End of the Project Gutenberg EBook of Romeo and Juliet,
by William Shakespeare
```

Eseguite un'ulteriore pulizia del testo, con un editor di testi, prima di procedere con l'analisi dell'opera.

- **Il nome di ogni personaggio viene menzionato tutte le volte che parla**, come di norma nelle opere teatrali. Per le analisi che farete in questo caso pratico, i nomi dei personaggi non servono, anzi sono d'intralcio. Manteneteli, invece, nel caso di rilevazioni della similarità più sofisticate.
- **Le opere teatrali includono anche molte indicazioni relative alla messa in scena, che segnalano quando i personaggi devono entrare o uscire di scena, combattere tra loro, cadere a terra e morire avvelenati, ecc.** Anche queste indicazioni vanno eliminate.

Potreste scrivere un programma che svolga queste operazioni di pulizia. Ma fate attenzione: infatti l'esame accurato del manoscritto potrebbe rivelare numerosi casi particolari che il vostro programma deve essere in grado di gestire. La scrittura di un programma che tenga conto di tutti questi casi potrebbe rivelarsi dispendiosa in termini di tempo e soggetta a errori.

Array necessari per eseguire l'analisi

Siete ora pronti ad analizzare *Romeo and Juliet* per produrre le statistiche che userete per un semplice rilevamento della similarità. Usate tre array per registrare i diversi tipi di conteggi che caratterizzano il testo dell'opera.

- L'array `sentenceLengths` tiene il conto di quante sono le frasi composte da una parola, quante da due parole, da tre parole, ecc.
- L'array `wordLengths` tiene il conto di quante sono le parole composte da un carattere, quante da due caratteri, da tre caratteri, ecc.
- L'array `wordFrequencies` contiene una struct per ogni singola parola presente nell'opera. I membri della struct sono la parola e il calcolo di quante volte la parola stessa compare nel testo. La parola viene memorizzata come una stringa, in un array di caratteri di lunghezza predeterminata, che deve essere grande abbastanza da contenere la parola più lunga dell'opera e il carattere nullo di terminazione.

Analisi di una frase

Prendiamo in esame la frase:

"O Romeo, Romeo, wherefore art thou Romeo."

- Questa frase è composta da sette parole, quindi il vostro programma deve aggiungere 1 all'array `sentenceLengths[7]`.
- La prima parola ("O") ha una sola lettera, quindi il vostro programma deve aggiungere 1 all'array `wordLengths[1]`.

- La seconda parola (“Romeo”) ha cinque lettere, quindi il vostro programma deve aggiungere 1 all’array `wordLengths[5]`, e così via.

Per eseguire il conteggio della frequenza delle parole, convertite le parole in lettere minuscole, in modo che tutte le occorrenze della stessa parola risultino uguali nel confronto. Quando elaborate ciascuna parola, ricercate nell’array `wordFrequencies` per verificare se la parola è già presente in esso. Se lo è, aggiungete uno al conteggio di questa parola; altrimenti, inserite la parola nel successivo elemento vuoto dell’array `wordFrequencies` e impostate il suo conteggio a 1.

Implementazione del codice di analisi

Utilizzate le tecniche di elaborazione di array, stringhe e file apprese in precedenza per leggere il contenuto di *Romeo and Juliet* ed eseguire i seguenti compiti.

- Ogni frase finisce con un **segno di punteggiatura terminale**: un **punto (.)**, un **punto di domanda (?)** o un **punto esclamativo (!)**. Definite una **funzione processSentence** che legge le parole fino a quando incontra un segno di punteggiatura terminale. Questa funzione aggiorna i contatori di frasi, parole e caratteri nel corso dell’elaborazione di ogni parola. Quando viene raggiunta la fine di una frase, incrementate il contatore appropriato nell’array `sentenceLengths` e reimpostate il contatore delle parole a zero.
- Per ciascuna parola, la funzione `processSentence` deve chiamare la **funzione processWord** per incrementare il contatore appropriato nell’array `wordLengths` e incrementare il contatore della parola nell’array `wordFrequencies` oppure aggiungere la parola all’array `wordFrequencies` con un contatore pari a 1.

Ricordate che dovete tenere traccia del numero totale di frasi, parole e caratteri.

Resoconto dell’analisi

Visualizzate poi le seguenti statistiche, relative all’analisi di *Romeo and Juliet*:

- il numero totale delle frasi;
- il numero totale delle parole;
- il numero totale dei caratteri;
- la lunghezza media delle frasi;
- la lunghezza media delle parole;
- la lunghezza mediana delle frasi;
- la lunghezza mediana delle parole;
- una tabella delle lunghezze delle frasi e della loro percentuale rispetto a tutte le lunghezze delle frasi;
- una tabella delle lunghezze delle parole e della loro percentuale rispetto a tutte le lunghezze delle parole;
- una tabella di distribuzione di frequenza contenente le singole parole dell’opera, la loro frequenza e percentuale rispetto al totale delle parole; visualizzatele in ordine decrescente di frequenza.

Il vostro programma deve inoltre salvare queste statistiche in un file, in modo che sia più semplice studiare i risultati e confrontarli con quelli di altre opere.

Analisi dell’opera Edward the Second di Christopher Marlowe

Dopo aver analizzato *Romeo and Juliet*, usate ora un editor di testi per la pulizia dell’opera *Edward the Second* di Christopher Marlowe. Una parte rilevante di tutti gli studi di data science è la conoscenza dei propri dati. Le convenzioni usate in *Edward the Second* per specificare quale personaggio sta parlando e per le indicazioni relative alla messa in scena sono diverse rispetto a quelle viste in *Romeo and Juliet*. Prestate quindi molta attenzione a queste differenze durante la pulizia di *Edward the Second*. Dopo aver pulito il testo, eseguite il vostro programma di analisi su *Edward the Second*. Confrontate i risultati con quelli prodotti per *Romeo and Juliet* e commentate le similarità tra le due opere.

Caso pratico su intelligenza artificiale/data science: machine learning con la Libreria Scientifica GNU

11.20 (Machine learning con regressione lineare semplice) Nell'ambito dell'intelligenza artificiale, uno dei campi più interessanti e promettenti è quello del machine learning. Il nostro obiettivo è di offrirvi qui una semplice introduzione pratica a una delle più elementari tecniche di machine learning.

Previsioni

Il machine learning viene comunemente usato per effettuare previsioni, basate su dati esistenti (spesso grandi quantità di dati). Non sarebbe fantastico poter ottimizzare le previsioni del tempo per riuscire a salvare vite umane, ridurre il numero dei feriti e i danni alle proprietà? E se potessimo fare progressi nella diagnosi e nel trattamento del cancro per salvare vite, o rendere più accurate le previsioni relative al business per massimizzare i profitti e garantire i posti di lavoro? Oppure identificare le frodi negli acquisti con carta di credito e nelle richieste di risarcimento alle assicurazioni? Prevedere l'abbandono (*churn*) dei clienti, i probabili prezzi di vendita delle case, gli incassi per i film in uscita e, più in generale, i ricavi attesi per nuovi prodotti e servizi? E ancora, individuare anticipatamente le strategie migliori che allenatori e giocatori possono usare per vincere più partite e campionati? Tutti questi tipi di previsioni oggi sono possibili grazie al machine learning.

Libreria Scientifica GNU e applicazione gnuplot

In questo caso pratico, dovete esaminare un programma con codice completo che mostra la tecnica di machine learning chiamata **regressione lineare semplice**, eseguita con una funzione della **Libreria Scientifica GNU** open source:

<https://www.gnu.org/software/gsl/>

Questa libreria definisce molti degli algoritmi comunemente usati in ingegneria, scienza e matematica. Il programma che studierete passa i comandi all'applicazione **gnuplot** per la creazione di grafici in 2D e 3D. Come vedrete, gnuplot usa un suo linguaggio diverso da quello del C per la creazione di grafici, quindi nel nostro codice abbiamo incluso commenti estesi che spiegano i comandi di gnuplot.

Statistiche descrittive

Nella data science, si usano spesso statistiche per descrivere e riepilogare i dati. Tra le **statistiche descrittive** di base troviamo:

- **minimo:** il valore più piccolo contenuto in una collezione di valori;
- **massimo:** il valore più grande contenuto in una collezione di valori;
- **intervallo:** la differenza tra il valore massimo e il valore minimo;
- **conteggio:** il numero di valori in una collezione;
- **somma:** il totale dei valori in una collezione.

Le **misure di dispersione** (chiamate anche **misure di variabilità**), come l'*intervallo*, aiutano a determinare quanto estesi siano i valori. Altre misure di dispersione sono la *varianza* e la *deviazione standard*.^{5,6,7}

Altre statistiche descrittive sono la media, la mediana e la moda, che abbiamo trattato nel Paragrafo 6.9. Queste sono **misure di tendenza centrale**: ognuna di esse produce un singolo valore che rappresenta il valore "centrale" in un insieme di valori, ovvero un valore in qualche modo rappresentativo degli altri.

Quartetto di Anscombe

Una fase fondamentale nell'analisi dei dati è "conoscere i propri dati". Le *statistiche descrittive di base* di cui sopra sono certamente utili nel comprendere meglio i propri dati. Fate attenzione, però, perché dataset totalmente diversi tra loro potrebbero in realtà avere statistiche descrittive identiche o quasi.

5. "Understanding Descriptive Statistics." Accesso 1 gennaio 2021. <https://towardsdatascience.com/understanding-descriptive-statistics-c9c2b0641291>.

6. "Standard deviation." Accesso 1 gennaio 2021. https://en.wikipedia.org/wiki/Standard_deviation.

7. "Variance." Accesso 1 gennaio 2021. <https://en.wikipedia.org/wiki/Variance>.

Un esempio di questo fenomeno viene presentato nel *quartetto di Anscombe*:

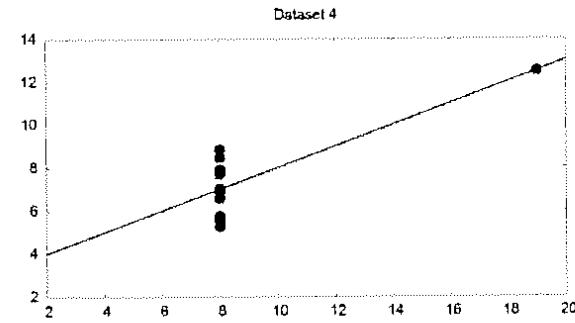
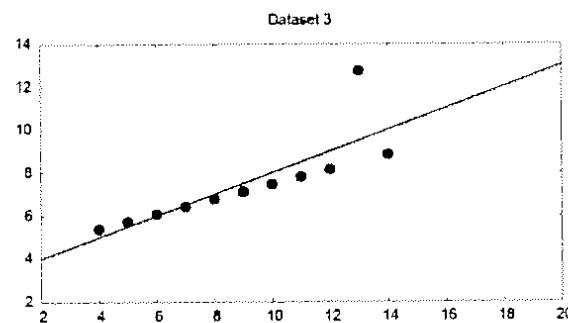
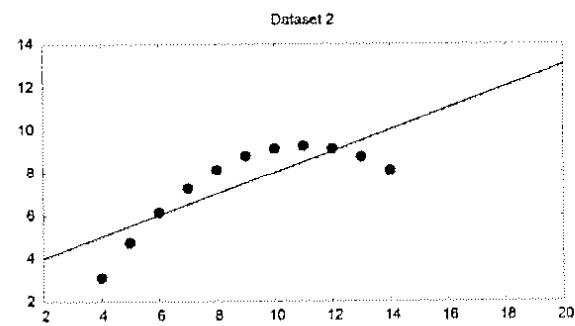
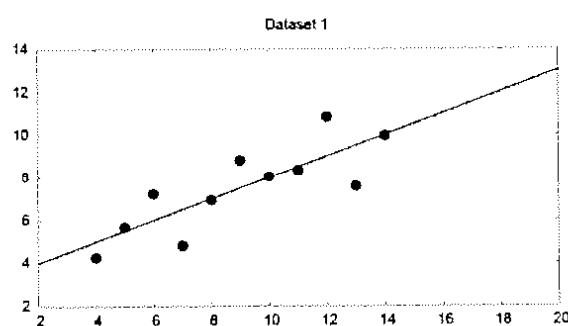
https://en.wikipedia.org/wiki/Anscombe%27s_quartet

Il quartetto di Anscombe comprende i seguenti quattro insiemi di coppie di coordinate x - y con 11 esempi di dati ciascuno:

x_1	y_1	x_2	y_2	x_3	y_3	x_4	y_4
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

È interessante notare che questi data set hanno statistiche descrittive *quasi identiche*. Per esempio, in tutti e quattro i dataset, i valori medi delle coordinate x e y sono rispettivamente 9 e 7,5.

I diagrammi seguenti, creati dal nostro **caso pratico con codice completo**, tracciano i dati, rispettivamente, di x_1 - y_1 , x_2 - y_2 , x_3 - y_3 e x_4 - y_4 :



Discuteremo a breve di queste lineerette (note come **rette di regressione**). Come si può vedere dalle **visualizzazioni** (ma non è detto che si riesca a vedere osservando semplicemente i dati nella tabella precedente) questi dataset del quartetto di Anscombe sono *significativamente diversi*. Eppure, come le loro statistiche descrittive, le loro rette di regressione risultano identiche. Ciò dimostra che non si possono trarre conclusioni solo dalle statistiche descrittive e dalle regressioni. Sono necessari ulteriori strumenti, come le visualizzazioni mostrate qui sopra, per **conoscere i propri dati**.

Regressione lineare semplice

Data una collezione di punti (coppie di coordinate x - y) che rappresentano una **variabile indipendente** (x) e una **variabile dipendente** (y), la **regressione lineare semplice** descrive la **relazione lineare** tra queste variabili con una linea retta, conosciuta come **retta di regressione**. Le rette nei diagrammi precedenti sono le rette di regressione per ciascuno dei quattro dataset del quartetto di Anscombe.

Consideriamo la relazione lineare tra le temperature in gradi Celsius e Fahrenheit. Data una temperatura in gradi Celsius, possiamo calcolare la temperatura corrispondente in gradi Fahrenheit usando la formula seguente:

$$\text{fahrenheit} = 9 / 5 * \text{celsius} + 32$$

In questa formula, *celsius* è la *variabile indipendente* e *fahrenheit* è la *variabile dipendente*. Ogni valore della temperatura *fahrenheit* *dipende* dal valore della temperatura *celsius* usato nel calcolo. Se si tracciano le temperature Fahrenheit per ciascuna temperatura Celsius, tutti i punti risultanti formano una linea retta, che mostra la *relazione lineare* tra le due scale di misura della temperatura.

Componenti nell'equazione di una regressione lineare semplice

I punti lungo qualsiasi linea retta (in due dimensioni) come le rette di regressione mostrate nei diagrammi precedenti, possono essere calcolati con l'equazione:

$$y = mx + b$$

dove

- m è il **coefficiente angolare** della retta (pendenza);
- b è l'**intercetta** della retta con l'asse y (quando $x = 0$), o semplicemente l'**intercetta y** ;
- x è la variabile indipendente;
- y è la variabile dipendente.

Nella formula per convertire le temperature Celsius in temperature Fahrenheit:

- m è $9 / 5$;
- b è 32 ;
- x è *celsius* (la variabile indipendente, ovvero la temperatura Celsius);
- y è *fahrenheit* (la variabile dipendente, ovvero la temperatura Fahrenheit risultante dal calcolo).

Nella regressione lineare semplice, y è il *valore previsto* a partire da un dato x . Dato che la linea retta ha un numero infinito di punti, se si riesce a determinare con una regressione lineare semplice l'equazione per una linea retta partendo da un numero limitato di punti campione, allora si ha la possibilità di fare un numero infinito di previsioni, anche per valori della variabile indipendente mai visti prima.

Funzionamento della regressione lineare semplice

La regressione lineare semplice è una tecnica di **machine learning** che determina la pendenza (m) e l'intercetta (b) della linea retta che meglio si adatta ai vostri dati. L'algoritmo della regressione lineare semplice aggiusta iterativamente la pendenza e l'intercetta, calcolando, a ogni aggiustamento, il quadrato della distanza di ogni punto dalla linea. La linea retta "più adatta" sarà quella con i valori di pendenza e intercetta che minimizzano la somma di queste distanze al quadrato. Questo è conosciuto come **metodo dei minimi quadrati**.⁸

8. https://en.wikipedia.org/wiki/Ordinary_least_squares.

Esecuzione della regressione lineare semplice con la Libreria Scientifica GNU

La funzione `gsl_fit_linear` della Libreria Scientifica GNU incapsula i calcoli della regressione lineare semplice, dando come risultati la pendenza e l'intercetta y della linea retta che meglio si adatta ai dati. Dopo la chiamata alla funzione `gsl_fit_linear`, basta inserire i valori della pendenza (m) e dell'intercetta (b) nell'equazione $y = mx + b$ per effettuare le previsioni dei valori dipendenti di y , basati sui valori indipendenti di x . Si possono poi usare questi valori con `gnuplot` per visualizzare la retta di regressione per i dati insieme ai punti dei dati.

File CSV

I dati relativi al quartetto di Anscombe sono contenuti nel file `anscombe.csv`. Questo file è il codice sorgente per l'esercizio sono nella sottocartella `AnscombesQuartet` nel Code Examples disponibile nella piattaforma MyLab. L'estensione del nome file `.csv` indica che il file è in formato **CSV (Comma-Separated Values, valori separati da virgole)**, un formato di file popolare specialmente per la distribuzione di dataset. I file CSV sono semplicemente file di testo nei quali ogni riga corrisponde a un record di informazioni i cui elementi sono separati da virgole. Le righe seguenti sono le prime due del file `anscombe.csv`:

```
x1,y1,x2,y2,x3,y3,x4,y4
10,8.04,10,9.14,10,7.46,8,6.58
```

La prima riga di un file CSV di norma contiene i nomi delle colonne per i dati nelle righe successive. Nelle altre righe del file `anscombe.csv` ci sono i valori numerici per le colonne. Nel nostro codice, la funzione `readAnscombesQuartetData` carica i dati negli array.

Installazione della Libreria Scientifica GNU su macOS

Si può installare la Libreria Scientifica GNU su macOS utilizzando il gestore di pacchetti **Homebrew**⁹ come segue:

```
brew install gsl
```

Installazione della Libreria Scientifica GNU su Windows

In Visual Studio, aggiungete la Libreria Scientifica GNU a ciascun progetto nel quale avete intenzione di utilizzarla. Aprite il vostro progetto in Visual Studio ed effettuate le seguenti operazioni:

1. selezionate **Tools > NuGet Package Manager > Manage NuGet Packages for Solution...**;
2. nella scheda **Browse**, ricercate "gsl-msvc-", poi selezionate `gsl-msvc-x64`;
3. nel lato destro del gestore di pacchetti NuGet, fate clic sulla casella vicino al nome del vostro progetto, quindi fate clic su **Install** per aggiungere la libreria al progetto.

Installazione di gnuplot su macOS

Installate il programma gnuplot utilizzando il gestore di pacchetti Homebrew come segue:

```
brew install gnuplot
```

Installazione di gnuplot su Windows

Scaricate ed eseguite il programma di installazione di gnuplot per Windows (`gp541-win64-mingw.exe`) dal sito:

<https://sourceforge.net/projects/gnuplot/files/gnuplot/5.4.1/>

Fate clic su **Next >** fino a raggiungere **Select Additional Tasks**, poi:

- sotto **Select gnuplot's default terminal**, selezionate il pulsante di opzione **windows**;
- scorrete fino in fondo alle impostazioni e spuntate la casella di controllo **Add application directory to your PATH environment variable**.

Fate clic su **Next >** e poi su **Install**. Quando l'installazione è completata, riavviate il computer.

9. Se non trovate il comando `brew`, consultate le istruzioni di installazione sul sito <https://brew.sh>.

Compilare ed eseguire il programma su macOS

Compilate anscome_macos.c su macOS effettuando le seguenti operazioni:

1. apriete una finestra di Terminale;
2. usate il comando cd per passare alla sottocartella AnscombesQuartet nel Code Examples disponibile nella piattaforma MyLab;
3. compilate il programma con il comando

```
clang -std=c18 anscombe_macos.c -lgsl -o anscombe_macos
```

4. eseguite il programma

```
./anscombe_macos
```

Il programma produrrà quattro file di immagini in formato PNG nella stessa cartella di anscome_macos.c su macOS. Aprite questi file per vedere i quattro grafici.

Compilare ed eseguire il programma su Windows

Nella soluzione di Visual Studio in cui avete aggiunto la Libreria Scientifica GNU:

- aggiungete al vostro progetto il file anscome_windows.c dalla sottocartella AnscombesQuartet nel Code Examples disponibile nella piattaforma MyLab;
- modificate la riga 72 in modo da specificare la posizione del file anscombe.csv nel vostro sistema;
- costruite ed eseguite il vostro progetto.

L'esecuzione del programma produrrà quattro file di immagini in formato PNG nella cartella del vostro progetto. Usate **Esplora file** per andare a quella cartella, quindi aprite i file delle immagini per vedere i quattro grafici.

Codice con commenti estesi

A seguire, studiate il codice per apprendere come usare la funzione gsl_fit_linear della Libreria Scientifica GNU per eseguire la regressione lineare semplice, e come inviare i comandi di gnuplot da un programma in C all'applicazione gnuplot. Provate a modificare i comandi di gnuplot per vedere gli effetti di questi cambiamenti sui grafici prodotti dal programma. Per esempio, modificate i valori del grafico relativi a tipo di punto, larghezza e colore della linea (pointtype, linewidth e linecolor).

Caso pratico su intelligenza artificiale/data science: serie temporali e regressione lineare semplice

Ora che avete studiato con attenzione il codice per il quartetto di Anscombe, potete adattare il programma per altri problemi di regressione lineare semplice. La regressione lineare semplice è usata comunemente per analizzare **serie temporali**, ovvero sequenze di valori (chiamati **osservazioni**) associati a punti nel tempo. Alcuni esempi: le chiusure giornaliere dei mercati, le rilevazioni orarie della temperatura, la posizione di un aereo durante il volo, la quantità annuale dei raccolti e i profitti trimestrali di un'azienda. Forse la serie temporale più significativa è il flusso di tweet con marca temporale provenienti da utenti Twitter di tutto il mondo.

Serie temporali

In questo esercizio userete la regressione lineare semplice per analizzare una serie temporale che contiene le temperature medie di gennaio a New York City *ordinate* per anno dal 1895 al 2020. Questa è una **serie temporale univariata** (contiene *una* osservazione alla volta). Una **serie temporale multivariata** ha *due o più* osservazioni alla volta, come le letture orarie di temperatura, umidità e pressione barometrica in un'applicazione meteorologica. Il vostro obiettivo in questo esercizio è quello di determinare se la retta di regressione ha:

- una **pendenza negativa**, che indica un **andamento della temperatura media in calo** nel corso di quel periodo di tempo;
- una **pendenza zero**, che indica un **andamento della temperatura media stabile** nel periodo;
- una **pendenza positiva**, che indica un **andamento della temperatura media in aumento** nel periodo.

Ottenere i dati climatici dal NOAA

La National Oceanic and Atmospheric Administration (NOAA)

<http://www.noaa.gov>

offre una grande quantità di dati climatici storici pubblici, incluse le serie temporali con le medie delle temperature in città specifiche in vari intervalli temporali.

Possiamo ottenere le temperature medie di gennaio a New York City dal 1895 al 2020 (l'intervallo massimo disponibile al momento della stesura di questo libro) dalle serie temporali "Climate at a Glance" del NOAA:

<https://www.ncdc.noaa.gov/cag/city/time-series>

È possibile selezionare i dati meteorologici per gli interi Stati Uniti, oppure per regioni, stati e città degli Stati Uniti, ecc. Dopo aver scelto i dati e l'intervallo di tempo che vi servono per l'analisi, fate clic su **Plot** per visualizzare un diagramma e una tabella dei dati selezionati. In cima alla tabella ci sono le icone su cui potete fare clic per scaricare i dati in diversi formati, tra cui il CSV.

Per comodità, vi forniamo il file `nyc_ave_january_temps.csv` contenente i dati da usare in questo esercizio. Il file si trova nella sottocartella `nycdata` nel Code Examples disponibile nella piattaforma MyLab. Abbiamo anche già "pulito" i dati, quindi il file contiene le seguenti due colonne per ogni osservazione.

- **Date:** un valore nel formato YYYY (come 2020). I dati scaricati sono nel formato YYYYMM (come 202001), dove 01 rappresenta gennaio. Abbiamo rimosso 01 da tutti i dati in questa colonna, lasciando solo l'anno.
- **Temperature:** una temperatura Fahrenheit con valori in virgola mobile. Abbiamo rinominato così la colonna chiamata **Value** nei dati scaricati.

Abbiamo rimosso una terza colonna chiamata **Anomaly** che non serve per questo esercizio.

Eseguire la regressione

Modificate il codice del quartetto di Ascombe per eseguire una regressione lineare semplice usando i dati delle temperature medie di gennaio a New York City e per disegnare un grafico dei dati con una retta di regressione. Quale andamento osservate negli ultimi 126 anni?

Caso pratico su servizi web e cloud: libcurl e OpenWeatherMap

11.21 (Ottenere il bollettino meteorologico di una città con OpenWeatherMap) Questo è un altro dei nostri esercizi impegnativi. Nel Paragrafo 1.11 abbiamo introdotto i concetti di **Internet**, **Web**, **cloud**, **servizi web** e **mashup**. In questo esercizio vi immergerete nel mondo dei servizi web usando le librerie open source **libcurl**¹⁰ e **cJSON**¹¹ per richiamare un servizio web ed elaborare i risultati ottenuti. Studierete un programma con codice completo e ampiamente commentato che interagisce con il servizio web gratuito **OpenWeatherMap** dal sito

<https://openweathermap.org/>

per ottenere il bollettino meteorologico attuale per una determinata città.

Successivamente, vi sfideremo a creare il vostro primo mashup con le conoscenze apprese da questo esempio con codice completo. Se avete iniziativa imprenditoriale, potete creare rapidamente prototipi di nuove applicazioni combinando servizi web esistenti in mashup. Anche se non affronterete la sfida di questo progetto, padroneggiare il codice di questo caso pratico vi renderà comunque in grado di accedere al vasto mondo dei servizi web.

Servizi web

La macchina su cui risiede un servizio web viene definita **host del servizio web**. Un'applicazione client (nel nostro caso, un programma C) invia una **richiesta** su una rete all'host del servizio web, che la elabora e restituisce una **risposta** al client tramite la rete. Questo tipo di calcolo distribuito avvantaggia i sistemi in vari modi. Per esempio, un'applicazione senza accesso diretto ai dati su un altro sistema potrebbe essere in grado di

10. "libcurl – the multiprotocol file transfer library." Accesso 4 gennaio 2021. <https://curl.se/libcurl/>.

11. "cJSON." Accesso 4 gennaio 2021. <https://github.com/DaveGamble/cJSON>.

recuperare i dati tramite un servizio web. Analogamente, un'applicazione che non abbia la potenza di elaborazione necessaria per eseguire calcoli specifici potrebbe utilizzare un servizio web per sfruttare risorse superiori di un altro sistema.

REST

La maggior parte dei servizi web attuali utilizzano uno stile architettonale noto come **REST** (*Representational State Transfer*) e vengono spesso chiamati **servizi web RESTful**. In un servizio web RESTful, ogni funzione che si può chiamare è identificata da un URL unico. Gli **URL** (*Uniform Resource Locators*) identificano la posizione in Internet di risorse quali siti e servizi web; quando un server web riceve una richiesta per un servizio web RESTful, sa subito quale funzione chiamare su quel server. I servizi web RESTful possono essere chiamati da programmi, come nel nostro caso, oppure direttamente dalla barra degli indirizzi di un browser web inserendo l'URL appropriato.

OpenWeatherMap

OpenWeatherMap mette a disposizione un livello gratuito per molti dei suoi servizi web meteo. Per poterli utilizzare, dovete registrarvi con un account gratuito sul sito <https://openweathermap.org/>. Vi verrà inviata un'e-mail di verifica; una volta confermato l'account, seguirà un'altra e-mail con la vostra chiave API gratuita, che troverete anche nella scheda **API Keys** accedendo al vostro account nel sito.

Potete vedere le varie API di OpenWeatherMap con la loro documentazione all'indirizzo:

<https://openweathermap.org/api>

Alcune sono gratuite, mentre altre sono disponibili solo agli abbonati al servizio. Con una chiave API gratuita, si può accedere a:

- **Current Weather Data:** i dati meteo attuali di una determinata località (verranno usati in questo caso pratico);
- **One Call API:** una chiamata API che restituisce una combinazione di dati meteo attuali e futuri di una determinata località;
- **5 Day/3 Hour Forecast:** previsioni meteo a 5 giorni con incrementi di 3 ore per una determinata località.

JSON

Molti servizi basati sul cloud, come OpenWeatherMap, comunicano con le applicazioni tramite oggetti JSON. **JSON** (*JavaScript Object Notation*) è un formato basato su testo per lo scambio di dati, leggibile da esseri umani e da computer, usato per rappresentare dati come collezioni di coppie nome/valore. JSON è diventato il formato preferito per la trasmissione di dati via Internet tra applicazioni diverse, in particolar modo per l'invocazione dei servizi web basati sul cloud.

Ogni oggetto JSON contiene una lista di *nomi di proprietà e valori* separati da virgolette, racchiusi tra parentesi graffe. Per esempio, le seguenti coppie chiave-valore potrebbero rappresentare il record di un cliente:

```
{"account": 100, "name": "Jones", "balance": 24.98}
```

JSON supporta anche gli array, che sono valori separati da virgolette racchiusi tra parentesi quadre. Per esempio, il seguente array di numeri è accettabile in JSON:

```
[100, 200, 300]
```

I valori negli oggetti e negli array JSON possono essere:

- stringhe tra *virgolette doppie* (come "Jones");
- numeri (come 100 o 24.98);
- valori booleani JSON (rappresentati come true o false);
- null (per rappresentare l'assenza di valore, come NULL in C);
- array (come [100, 200, 300]);
- altri oggetti JSON.

La Figura 11.8 contiene un esempio di risposta in JSON dal servizio web Current Weather Data di OpenWeatherMap, da noi formattato per leggibilità. Anche senza aver mai visto prima dati codificati in formato JSON, vi potete rendere conto da questo esempio di come siano ben organizzati, leggibili e di facile comprensione.

```

1  {
2    "coord": {
3      "lon": -71.06,
4      "lat": 42.36
5    },
6    "weather": [
7      {
8        "id": 803,
9        "main": "Clouds",
10       "description": "broken clouds",
11       "icon": "04n"
12     }
13   ],
14   "base": "stations",
15   "main": {
16     "temp": 0.03,
17     "feels_like": -4.96,
18     "temp_min": -1.11,
19     "temp_max": 1.11,
20     "pressure": 1014,
21     "humidity": 93
22   },
23   "visibility": 10000,
24   "wind": {
25     "speed": 4.1,
26     "deg": 360
27   },
28   "clouds": {
29     "all": 75
30   },
31   "dt": 1609815037,
32   "sys": {
33     "type": 1,
34     "id": 3486,
35     "country": "US",
36     "sunrise": 1609762409,
37     "sunset": 1609795488
38   },
39   "timezone": -18000,
40   "id": 4930956,
41   "name": "Boston",
42   "cod": 200
43 }
```

Figura 11.8 Esempio di risposta da OpenWeatherMap per Boston, MA, USA.

Libreria open source libcurl

Per ottenere la risposta in JSON della Figura 11.8, la nostra applicazione usa le funzioni della libreria open source libcurl:

<https://curl.se/libcurl/>

La libreria supporta molti dei protocolli Internet e web per la trasmissione di dati tra applicazioni e può essere usata per invocare servizi web e ricevere le relative risposte. La documentazione per le funzioni C di libcurl è all'indirizzo:

```
https://curl.se/libcurl/c/
```

Nel nostro esempio con codice completo, `weather.c`, contenuto nella sottocartella `weather` nel Code Examples disponibile nella piattaforma MyLab, abbiamo inserito ampi commenti sulle funzioni di libcurl necessarie per invocare un servizio web e salvare la relativa risposta in un file.

Per installare libcurl su macOS o Linux:

- su macOS, installate la libreria libcurl utilizzando il gestore di pacchetti Homebrew¹² come segue:

```
brew install libcurl4
```

- su Ubuntu Linux, eseguite il comando:

```
sudo apt install libcurl4-openssl-dev
```

In Visual Studio, aggiungete la libreria libcurl a ciascun progetto nel quale avete intenzione di utilizzarla. Aprirete il vostro progetto in Visual Studio ed effettuate le seguenti operazioni.

1. Selezionate **Tools > NuGet Package Manager > Manage NuGet Packages for Solution....**
2. Nella scheda **Browse**, cercate "curl", poi selezionate "curl by curl contributors".
3. Nel lato destro del **NuGet Package Manager**, fate clic sulla casella di controllo vicino al nome del vostro progetto, quindi fate clic su **Install** per aggiungere la libreria al progetto.

Libreria open source cJSON

La parte libcurl della nostra applicazione scrive in un file la risposta in JSON del servizio OpenWeatherMap. Per visualizzare il bollettino meteo, la nostra app salva il contenuto del file in una stringa, quindi usa la **libreria cJSON** open source per estrarre gli elementi dal JSON. Si può scaricare la libreria cJSON da:

```
https://github.com/DaveGamble/cJSON
```

Non esiste una procedura di installazione per questa libreria. Includevi semplicemente nel vostro progetto i file `cJSON.h` e `cJSON.c` della libreria.

Le funzioni di cJSON abilitano l'accesso agli elementi nella risposta in JSON, quindi possiamo visualizzare un bollettino meteo come il seguente:

```
Boston Weather
Temperature: 0.0 °C
Feels like: -5.0 °C
Pressure: 1014 hPa
Humidity: 93%
Conditions: broken clouds
```

In `weather.c`, abbiamo inserito ampi commenti sulle funzioni di cJSON necessarie per estrarre i dati elencati sopra relativi alla città specificata quando si esegue l'applicazione (vedi il paragrafo successivo).

Compilare ed eseguire il programma su macOS e Linux

Su macOS e Linux, compilate l'applicazione meteo effettuando i seguenti passaggi.

1. Aprite una finestra di Terminale.
2. Usate il comando `cd` per passare alla sottocartella `weather` nel Code Examples disponibile nella piattaforma MyLab.

12. Se non trovate il comando `brew`, consultate le istruzioni di installazione sul sito <https://brew.sh/>.

3. Compilate il programma con uno dei seguenti comandi (`clang` su macOS o `gcc` su Linux):

```
clang -std=c18 -Wall weather.c cJSON.c -lcurl -o weather
gcc -std=c18 -Wall weather.c cJSON.c -lcurl -o weather
```

Questa applicazione riceve due argomenti della riga di comando. Gli argomenti della riga di comando non verranno trattati dettagliatamente sino al Paragrafo 15.3, ma in questa simulazione con codice completo trovate tutte le istruzioni necessarie per ricevere gli argomenti della riga di comando. Il primo è la città per la quale volete ottenere i dati meteo correnti, come:

`Boston,MA,USA`

Se il nome della città contiene uno spazio, racchiudete la località tra virgolette:

`"Los Angeles,CA,USA"`

Il secondo argomento della riga di comando è la vostra chiave API di OpenWeatherMap. Il comando seguente ottiene i dati meteo attuali per Boston, MA, USA:

```
./weather Boston,MA,USA API_KEY
```

Assicuratevi di sostituire `API_KEY` con la chiave API di OpenWeatherMap che avete ricevuto quando vi siete registrati per il vostro account gratuito.

Compilate il programma ed eseguitelo diverse volte. In seguito, studiate il codice di questa applicazione (inclusi gli ampi commenti).

Compilare ed eseguire l'applicazione meteo in Visual Studio

Nella soluzione di Visual Studio in cui avete aggiunto la libreria `libcurl`, aggiungete al vostro progetto i file `weather.c` e `cJSON.c` dalla sottocartella `weather` nel Code Examples disponibile nella piattaforma MyLab. Specificate gli argomenti della riga di comando come segue:

- fate clic con il tasto destro del mouse sul nome del progetto nella finestra Solution Explorer e selezionate **Properties**;
- espandete **Configuration Properties** e selezionate **Debugging**;
- inserite gli argomenti nella casella di testo a destra di **Command Arguments**;
- costruite ed eseguite il vostro progetto.

Sfida: creare il proprio mashup

I mashup sono stati introdotti nel Paragrafo 1.11.3. Una volta studiato il codice della nostra applicazione meteo basata su `libcurl` e i servizi web, inclusi gli ampi commenti, vi proponiamo un esercizio impegnativo: provate a creare il vostro primo mashup. I mashup web solitamente combinano le funzionalità di due o più servizi web complementari. In molti mashup popolari, uno di questi servizi è un servizio di mappatura, come Google Maps o Bing Maps di Microsoft, ma ci sono molte altre possibilità di mashup che non usano questo tipo di servizio.

Per costruire un mashup di servizi web, in genere occorre:

- identificare due o più servizi web complementari, che combinati possano produrre una nuova applicazione interessante;
- essere in grado di inviare una richiesta dal vostro programma in C a un servizio web, come avete imparato a fare con `libcurl` in questo caso pratico;
- essere in grado di ricevere i risultati da quel servizio web in una forma (tipicamente JSON) comprensibile per il vostro programma C.

La directory di servizi web ProgrammableWeb

<https://programmableweb.com/>

contiene un elenco di quasi 24.000 servizi web e 8.000 mashup. Vi sono anche istruzioni ed esempi di codice per lavorare con servizi web e creare i vostri mashup. Secondo il loro sito, tra i servizi web più utilizzati vi sono Google Maps e altri forniti da Facebook, Twitter e YouTube.

Prendete confidenza con ProgrammableWeb. Analizzate un buon numero dei servizi web descritti, concentrandovi su quelli gratuiti (è pratica comune per i fornitori di servizi web offrire alcuni servizi gratuiti e altri a pagamento). Leggete inoltre le istruzioni di ProgrammableWeb per creare i vostri mashup. Prendete ispirazione scorrendo tra gli 8.000 mashup presenti nel sito. Cercate di trovare due servizi web complementari da cui poter creare un'applicazione interessante, poi costruite il vostro mashup.

CAPITOLO

12

Sommario del capitolo

- 12.1 Introduzione
- 12.2 Strutture autoreferenziali
- 12.3 Gestione dinamica della memoria
- 12.4 Liste collegate
- 12.5 Pile
- 12.6 Code
- 12.7 Alberi
- 12.8 Programmazione sicura in C
- 12.9 Riepilogo

Strutture di dati

Obiettivi

- Allocare e liberare memoria in modo dinamico per i dati.
- Formare strutture di dati collegate usando puntatori, strutture autoreferenziali e ricorsione.
- Creare e manipolare liste collegate, code, pile e alberi binari.
- Conoscere importanti applicazioni delle strutture di dati collegate.
- Studiare le raccomandazioni per la programmazione sicura in C per puntatori e allocazione dinamica della memoria.
- Costruire facoltativamente un compilatore personale negli esercizi.

12.1 Introduzione

Abbiamo studiato le strutture di dati di dimensione fissa come gli array unidimensionali, gli array bidimensionali e le strutture (`struct`). Questo capitolo introduce le **strutture dinamiche di dati** che possono crescere e ridursi al momento dell'esecuzione.

- Le **liste collegate** sono collezioni di elementi di dati “allineati in una riga”. Potete inserire ed eliminare elementi in qualsiasi punto di una lista collegata.
- Le **pile** sono importanti nei compilatori e nei sistemi operativi. Potete inserire ed eliminare elementi solo a un estremo della pila, noto come **cima**.
- Le **code** rappresentano le file di attesa. Potete inserire elementi solo alla fine della coda (**tail**) ed eliminarne solo dalla testa della coda (**head**).
- Gli **alberi binari** facilitano la ricerca e l'ordinamento dei dati ad alta velocità, l'efficiente eliminazione dei dati duplicati e la compilazione di espressioni nel linguaggio macchina.

Ciascuna di queste strutture di dati ha molte altre applicazioni interessanti.

Progetto facoltativo: costruire il proprio compilatore

Ci auguriamo che tenterete il progetto più importante facoltativo descritto nel paragrafo speciale relativo alla costruzione di un compilatore alla fine degli esercizi. Finora avete continuato a usare un compilatore per tradurre i vostri programmi in C nel linguaggio macchina in modo da poterli eseguire. In questo progetto creerete effettivamente il vostro com-

pilatore. Questo leggerà un file di istruzioni scritte in un linguaggio semplice ma formidabile, di alto livello. Il vostro compilatore tradurrà queste istruzioni in un file di istruzioni del linguaggio macchina del Simpletron (SML). SML è il linguaggio (creato da Deitel) che avete imparato nel “Paragrafo speciale: costruite il vostro computer come macchina virtuale” del Capitolo 7. Il vostro programma del simulatore Simpletron eseguirà quindi il programma in SML prodotto dal vostro compilatore! Questo progetto vi darà la splendida opportunità di esercitarvi su quasi tutto quello che avete appreso nel corso del libro. Il paragrafo speciale vi mostrerà in dettaglio come specificare il linguaggio di alto livello e descriverà gli algoritmi di cui avrete bisogno per convertire ogni tipo di istruzione del linguaggio di alto livello nelle istruzioni del linguaggio macchina. Se vi diverte la sfida, potrete tentare i numerosi miglioramenti sia del compilatore sia del simulatore Simpletron consigliati negli esercizi.

✓ Autovalutazione

1. (*Completare*) Quale struttura di dati è descritta con “facilita la ricerca e l’ordinamento dei dati ad alta velocità, l’efficiente eliminazione dei dati duplicati e la compilazione di espressioni nel linguaggio macchina”? _____.

Risposta: L’albero binario.

2. (*Completare*) Quale struttura di dati è descritta con “una collezione di elementi di dati ‘allineati in una riga’; inserimenti ed eliminazioni possono essere fatti ovunque nella struttura di dati”? _____.

Risposta: La lista collegata.

3. (*Completare*) Quale struttura di dati dinamica è descritta con “potete inserire ed eliminare elementi solo a un suo estremo, noto come cima”? _____.

Risposta: La pila.

12.2 Strutture autoreferenziali

Una struttura autoreferenziale contiene un membro puntatore che punta a una struttura dello stesso tipo. Per esempio, la seguente definizione crea il tipo `struct node`:

```
struct node {
    int data;
    struct node *nextPtr;
};
```

La nostra struttura `struct node` ha due membri: il membro intero `data` e il membro puntatore `nextPtr`. Il membro `nextPtr` punta a un’altra struttura `struct node`, che è dello stesso tipo di quella che stiamo definendo, da qui il termine struttura autoreferenziale. Il membro `nextPtr` è chiamato **link** (collegamento) e può essere usato per “legare” una struttura di tipo `struct node` a un’altra struttura dello stesso tipo. Le strutture autoreferenziali possono essere collegate insieme per formare liste, code, pile e alberi.

Il seguente diagramma illustra due oggetti aventi come tipo una struttura autoreferenziale collegati insieme per formare una lista:



La barra¹ nell’ultimo nodo rappresenta un puntatore `NULL`, a indicare che il nodo non punta a un altro nodo. Un puntatore `NULL` indica la fine di una struttura di dati. Non impostare il link nell’ultimo nodo di una lista a `NULL` può portare a errori in fase di esecuzione.



1. La barra è solo per fini illustrativi e non corrisponde al carattere backslash del C.

✓ Autovalutazione

1. (*Completare*) Cosa dovrebbe sostituire ??? nel seguente codice per rendere questa una struttura (*struct*) autoreferenziale? _____.

```
struct node {
    int data;
    ??? *nextPtr;
};
```

Risposta: struct node.

2. (*Completare*) Un puntatore _____ indica la fine di una struttura di dati.

Risposta: NULL.

3. (*Completare*) Le strutture autoreferenziali possono essere _____ insieme per formare strutture di dati come liste, code, pile e alberi.

Risposta: collegate.

12.3 Gestione dinamica della memoria

Creare e mantenere strutture di dati dinamiche che possono crescere e ridursi durante l'esecuzione del programma richiede la **gestione dinamica della memoria**, che ha due componenti:

- ottenere un maggiore spazio di memoria al momento dell'esecuzione per contenere nuovi nodi;
- liberare spazio non più necessario.

La funzione `malloc`, la funzione `free` e l'operatore `sizeof` sono essenziali per la gestione dinamica della memoria.

Funzione malloc

Per richiedere memoria in fase di esecuzione, passate alla funzione `malloc` il numero di byte da allocare. In caso di successo, `malloc` restituisce un puntatore `void *` alla memoria allocata. Ricordatevi che un puntatore `void *` può essere assegnato a una variabile di un *qualsiasi* tipo di puntatore.

La funzione `malloc` viene usata normalmente con l'operatore `sizeof`. Per esempio, la seguente istruzione determina la dimensione in byte di un oggetto `struct node` con `sizeof(struct node)`, alloca una nuova area nella memoria di quel numero di byte e memorizza un puntatore alla memoria allocata in `newPtr`:

```
newPtr = malloc(sizeof(struct node));
```

Non è garantito che la memoria sia inizializzata, sebbene molte implementazioni la inizializzino per sicurezza. Se non è disponibile alcuna memoria, `malloc` restituisce `NULL`. Verificate sempre la presenza di un puntatore `NULL` prima di accedere alla memoria allocata dinamicamente per evitare errori in fase di esecuzione che potrebbero arrestare il vostro programma.

Funzione free

Quando non avete più bisogno di un blocco di memoria allocata dinamicamente, restituitelo al sistema immediatamente chiamando la funzione `free` per deallocare la memoria. Per liberare la memoria della precedente chiamata a `malloc` usate l'istruzione

```
free(newPtr);
```

Dopo questa operazione, impostate il puntatore a `NULL`: in questo modo evitate che il programma faccia riferimento alla memoria che è stata liberata e che potrebbe essere già stata allocata per un altro scopo.

 Non liberare memoria allocata dinamicamente quando non è più necessaria può far sì che il sistema esaurisca prematuramente la memoria. Ciò viene talvolta indicato con il termine "memory leak" ("falla nella memoria"). Fare riferimento alla memoria che è stata liberata è un errore che tipicamente provoca l'arresto anomalo del programma. Liberare la memoria non allocata in modo dinamico con `malloc` è un errore.

Funzioni calloc e realloc

Il C fornisce le funzioni `calloc` e `realloc` per creare e modificare la dimensione di array dinamici. Queste funzioni sono analizzate nel Paragrafo 15.8.

✓ Autovalutazione

1. (*Vero/Falso*) La funzione `malloc` riceve come argomento il numero di byte da allocare e restituisce un puntatore `NULL`.

Risposta: *Falso*. In realtà, `malloc` restituisce un puntatore `void *` con l'indirizzo della memoria allocata o restituisce un puntatore `NULL` se la memoria non può essere allocata.

2. (*Discussione*) Descrivete cosa fa esattamente l'istruzione seguente:

```
newPtr = malloc(sizeof(struct node));
```

Risposta: L'istruzione valuta `sizeof(struct node)` per determinare la dimensione dell'oggetto in byte, alloca nella memoria una nuova area con quel numero di byte e memorizza un puntatore alla memoria allocata in `newPtr`.

3. (*Discussione*) Scrivete un'istruzione per liberare la memoria già allocata dinamicamente a `newPtr` da `malloc`.

Risposta: `free(newPtr);`

4. (*Completare*) Non usare `free` per restituire la memoria allocata in modo dinamico quando non serve più può far sì che il sistema esaurisca la memoria. Ciò viene talvolta chiamato un “_____”.

Risposta: memory leak.

12.4 Liste collegate

Una **lista collegata** è una collezione con organizzazione lineare di oggetti `struct` autoreferenziali, chiamati **nodi**, connessi da collegamenti di puntatori, da cui il termine lista “collegata”. Si accede a una lista collegata tramite un puntatore al suo primo nodo e si accede ai nodi successivi tramite il puntatore di collegamento memorizzato in ogni nodo. I dati sono memorizzati dinamicamente in una lista collegata creando ogni nodo quando necessario. Un nodo può contenere dati di ogni tipo compresi altri oggetti `struct`. Anche le pile e le code sono **strutture lineari di dati** e, come vedremo, sono versioni vincolate di liste collegate.

Array vs liste collegate

Le liste di dati possono essere memorizzate in array, ma le liste collegate presentano diversi vantaggi.



- Una lista collegata è appropriata quando il numero di elementi da rappresentare nella struttura di dati non è prevedibile. Le liste collegate sono dinamiche, pertanto la lunghezza di una lista può aumentare o diminuire quando necessario. Gli array sono strutture di dati di dimensione fissa (sebbene il Paragrafo 15.8 mostri come allocare e riallocare dinamicamente gli array).
- Un array può essere dichiarato con più elementi rispetto a quelli attesi, ma ciò può comportare uno spreco di memoria. L'uso di liste collegate e dell'allocazione dinamica della memoria per le strutture di dati che crescono e si riducono al momento dell'esecuzione può far risparmiare memoria. Tenete a mente, comunque, che i puntatori nei nodi di una lista richiedono memoria aggiuntiva. Inoltre, l'allocazione dinamica della memoria espone al rischio di un sovraccarico di chiamate a funzioni.
- Gli array di dimensione fissa possono riempirsi. Le liste collegate si riempiono solo quando il sistema non ha *memoria* sufficiente per soddisfare le richieste di allocazione dinamica della memoria.
- Le liste collegate possono essere mantenute in ordine inserendo ogni nuovo elemento nel punto giusto nella lista. Gli inserimenti e le cancellazioni di dati in un array ordinato richiedono tempo. Tutti gli elementi che seguono l'elemento inserito o cancellato devono essere opportunamente spostati.

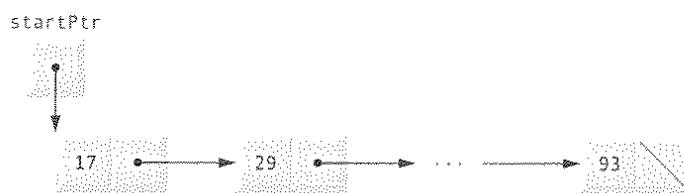


Accesso immediato agli elementi di un array

D'altra parte, gli elementi di un array sono registrati nella memoria in modo contiguo. Questo permette l'accesso immediato a un elemento di un array, perché l'indirizzo dell'elemento può essere calcolato direttamente in base alla sua posizione rispetto all'inizio dell'array. Le liste collegate non offrono un tale accesso immediato ai loro elementi.

Illustrare una lista collegata

Non è garantito che i nodi delle liste collegate siano registrati in modo contiguo nella memoria. Dal punto di vista logico, tuttavia, i nodi appaiono contigui. Il seguente diagramma illustra una lista collegata con diversi nodi.



Implementare una lista collegata

Il programma della Figura 12.1 manipola una lista di caratteri. È possibile inserire un carattere nella lista in ordine alfabetico (funzione `insert`) o eliminare un carattere dalla lista (funzione `delete`). Segue un'analisi dettagliata del programma. Abbiamo suddiviso il codice di questo programma ai fini dell'analisi; l'output è mostrato alla fine della prima tabella di codice di seguito.

```

1 // fig12_01.c
2 // Inserimento ed eliminazione di nodi in una lista
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 // struttura autoreferenziale
7 struct listNode {
8     char data; // ogni listNode contiene un carattere
9     struct listNode *nextPtr; // puntatore al nodo successivo
10 };
11
12 typedef struct listNode ListNode; // sinonimo per struct listNode
13 typedef ListNode *ListNodePtr; // sinonimo per ListNode*
14
15 // prototipi
16 void insert(ListNodePtr *sPtr, char value);
17 char delete(ListNodePtr *sPtr, char value);
18 int isEmpty(ListNodePtr sPtr);
19 void printList(ListNodePtr currentPtr);
20 void instructions(void);
21
22 int main(void) {
23     ListNodePtr startPtr = NULL; // inizialmente non vi sono nodi
24     char item = '\0'; // char inserito dall'utente
25
26     instructions(); // stampa il menu
27     printf("%s", "? ");
28     int choice = 0; // scelta dell'utente
29     scanf("%d", &choice);
  
```

```
30 // ripeti il ciclo finche' l'utente non sceglie 3
31 while (choice != 3) {
32     switch (choice) {
33         case 1: // inserisci un elemento
34             printf("%s", "Enter a character: ");
35             scanf("\n%c", &item);
36             insert(&startPtr, item); // inserisci l'elemento nella lista
37             printList(startPtr);
38             break;
39         case 2: // cancella un elemento
40             if (!isEmpty(startPtr)) { // se la lista non e' vuota
41                 printf("%s", "Enter character to be deleted: ");
42                 scanf("\n%c", &item);
43
44                 // se il carattere viene trovato, rimuovilo
45                 if (delete(&startPtr, item)) { // rimuovi l'elemento
46                     printf("%c deleted.\n", item);
47                     printList(startPtr);
48                 }
49             } else {
50                 printf("%c not found.\n\n", item);
51             }
52         }
53     }
54     else {
55         puts("List is empty.\n");
56     }
57
58     break;
59     default:
60         puts("Invalid choice.\n");
61         instructions();
62         break;
63     }
64
65     printf("%s", "? ");
66     scanf("%d", &choice);
67 } // fine di while
68
69     puts("End of run.");
70 }
71
72 // stampa le istruzioni del programma per l'utente
73 void instructions(void) {
74     puts("Enter your choice:\n"
75         "    1 to insert an element into the list.\n"
76         "    2 to delete an element from the list.\n"
77         "    3 to end.");
78 }
79
```

```

Enter your choice:
 1 to insert an element into the list.
 2 to delete an element from the list.
 3 to end.

? 1
Enter a character: B
The list is:
B --> NULL

? 1
Enter a character: A
The list is:
A --> NULL

? 1
Enter a character: C
The list is:
A --> B --> C --> NULL

? 2
Enter character to be deleted: D
D not found.

? 2
Enter character to be deleted: B
B deleted.

The list is:
A --> C --> NULL

? 2
Enter character to be deleted: C
C deleted.

The list is:
A --> NULL

? 2
Enter character to be deleted: A
A deleted.

List is empty.

? 4
Invalid choice.

Enter your choice:
 1 to insert an element into the list.
 2 to delete an element from the list.
 3 to end.

? 3
End of run.

```

Figura 12.1 Inserimento ed eliminazione di nodi in una lista.

Le righe 7-10 definiscono la struttura autoreferenziale `struct listNode`, che usiamo per costruire la lista collegata di questo esempio. Le righe 12 e 13 definiscono `typedef`, per rendere il codice più leggibile. Il nome `ListNode` rappresenta un oggetto `struct listNode`, e il nome `ListNodePtr` rappresenta un puntatore a un oggetto `struct listNode`. La funzione `main` consente l'inserimento di caratteri nella lista (righe 34-39), cancella elementi dalla lista (righe 40-58) o termina il programma. Inizialmente, `startPtr` (riga 23) è impostato a `NULL` per indicare una lista vuota. Le funzioni principali delle liste collegate del programma sono `insert` (Paragrafo 12.4.1) e `delete` (Paragrafo 12.4.2).

12.4.1 Funzione insert

Per questo esempio, abbiamo inserito i caratteri nella lista in ordine alfabetico. La funzione `insert` (righe 81-110) riceve come argomento l'*indirizzo del puntatore al primo nodo della lista* e un carattere da inserire. Ciò permette a `insert` di modificare il puntatore al primo nodo della lista della funzione chiamante, in modo che punti a un nuovo primo nodo quando un elemento dei dati viene inserito all'inizio della lista. Quindi passiamo il *puntatore* per riferimento. Passare l'indirizzo di un puntatore crea un **puntatore a un puntatore** (questo viene a volte chiamato *indirezione doppia*). Si tratta di una nozione complessa che richiede una programmazione accurata.

```

80 // inserisci un nuovo valore nella lista ordinata
81 void insert(ListNodePtr *sPtr, char value) {
82     ListNodePtr newPtr = malloc(sizeof(ListNode)); // crea il nodo
83
84     if (newPtr != NULL) { // c'e' spazio disponibile?
85         newPtr->data = value; // inserisci value nel nodo
86         newPtr->nextPtr = NULL; // il nodo non e' collegato ad altri nodi
87
88         ListNodePtr previousPtr = NULL;
89         ListNodePtr currentPtr = *sPtr;
90
91         // ripeti il ciclo per trovare la posizione corretta nella lista
92         while (currentPtr != NULL && value > currentPtr->data) {
93             previousPtr = currentPtr; // va avanti ...
94             currentPtr = currentPtr->nextPtr; // ... al nodo successivo
95         }
96
97         // inserisci il nuovo nodo all'inizio della lista
98         if (previousPtr == NULL) {
99             newPtr->nextPtr = *sPtr;
100            *sPtr = newPtr;
101        }
102        else { // inserisci il nuovo nodo tra previousPtr e currentPtr
103            previousPtr->nextPtr = newPtr;
104            newPtr->nextPtr = currentPtr;
105        }
106    }
107    else {
108        printf("%c not inserted. No memory available.\n", value);
109    }
110 }
111

```

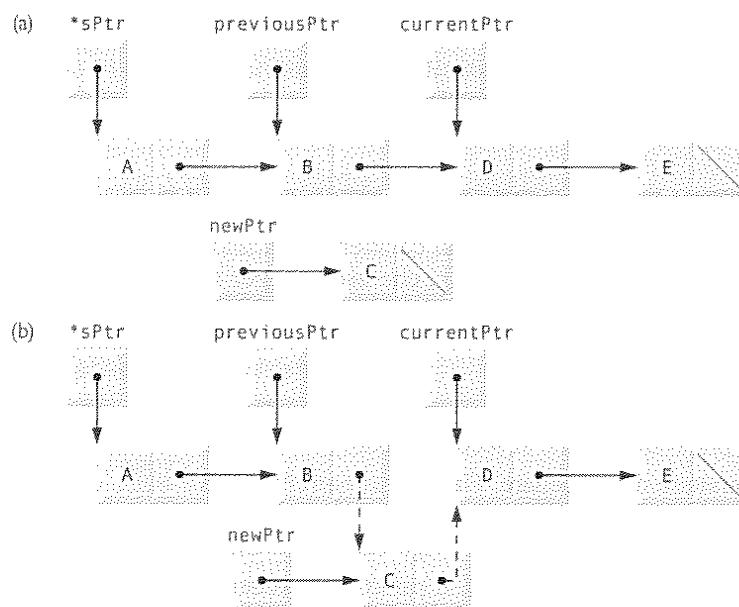
La funzione `insert` effettua i seguenti passi.

- Chiamate `malloc` per creare un nuovo nodo e assegnare a `newPtr` l'indirizzo di memoria allocata (riga 82).
- Se la memoria era stata allocata, assegnate il carattere da inserire a `newPtr->data` (riga 85), e assegnate `NULL` a `newPtr->nextPtr` (riga 86). Assegnate sempre `NULL` al membro `link` di un nuovo nodo. I puntatori devono essere inizializzati prima di essere usati.
- Useremo i puntatori `previousPtr` e `currentPtr` per memorizzare rispettivamente le posizioni del nodo che *precede* e che *segue* il punto d'inserzione. Iniziate `previousPtr` a `NULL` (riga 88) e `currentPtr` a `*sPtr` (riga 89), ovvero l'indirizzo del primo nodo.
- Localizzate il punto d'inserzione del nuovo valore. Finché `currentPtr` non è `NULL` e il valore da inserire è maggiore di `currentPtr->data` (riga 92), assegnate `currentPtr` a `previousPtr` (riga 93), quindi fate avanzare `currentPtr` al nodo successivo della lista (riga 94).
- Inserite il nuovo valore nella lista. Se `previousPtr` è `NULL` (riga 98), inserite il nuovo nodo come *primo* nella lista (righe 99-100). Assegnate `*sPtr` a `newPtr->nextPtr` (il collegamento del nuovo nodo punta al nodo che era *primo* in precedenza), e assegnate `newPtr` a `*sPtr`, così `startPtr` in `main` punta ora al nuovo *primo* nodo. Altrimenti, inserite il nuovo nodo nella lista (righe 103-104). Assegnate `newPtr` a `previousPtr->nextPtr` (il nodo precedente punta al nuovo nodo) e assegna `currentPtr` a `newPtr->nextPtr` (il collegamento del *nuovo* nodo punta al nodo *corrente*).

Per semplicità, abbiamo implementato la funzione `insert` (e altre funzioni simili in questo capitolo) con un tipo di ritorno `void`. È possibile che la funzione `malloc` *non riesca* ad allocare la memoria richiesta. In questo caso, sarebbe meglio che la nostra funzione `insert` restituisse un valore che indichi se l'operazione ha avuto successo.

Illustrare un inserimento

Il seguente diagramma illustra l'inserimento di un nodo contenente 'C' in una lista ordinata. La parte (a) della figura mostra la lista e il nuovo nodo appena prima dell'inserimento. La parte (b) della figura mostra il risultato dell'inserimento del nuovo nodo. Le frecce tratteggiate rappresentano i puntatori riassegnati.



12.4.2 Funzione delete

La funzione `delete` (righe 113-141) riceve l'indirizzo del puntatore al primo nodo della lista e un carattere da eliminare.

```

112 // elimina un elemento della lista
113 char delete(ListNodePtr *sPtr, char value) {
114     // elimina il primo nodo se viene trovata una corrispondenza
115     if (value == (*sPtr)->data) {
116         ListNodePtr tempPtr = *sPtr; // aggancia il nodo da rimuovere
117         *sPtr = (*sPtr)->nextPtr; // sfila il nodo
118         free(tempPtr); // libera il nodo sfilato
119         return value;
120     }
121     else {
122         ListNodePtr previousPtr = *sPtr;
123         ListNodePtr currentPtr = (*sPtr)->nextPtr;
124
125         // ripeti il ciclo per trovare la posizione corretta nella lista
126         while (currentPtr != NULL && currentPtr->data != value) {
127             previousPtr = currentPtr; // va avanti ...
128             currentPtr = currentPtr->nextPtr; // ... al nodo successivo
129         }
130
131         // elimina il nodo a currentPtr
132         if (currentPtr != NULL) {
133             ListNodePtr tempPtr = currentPtr;
134             previousPtr->nextPtr = currentPtr->nextPtr;
135             free(tempPtr);
136             return value;
137         }
138     }
139
140     return '\0';
141 }
142

```

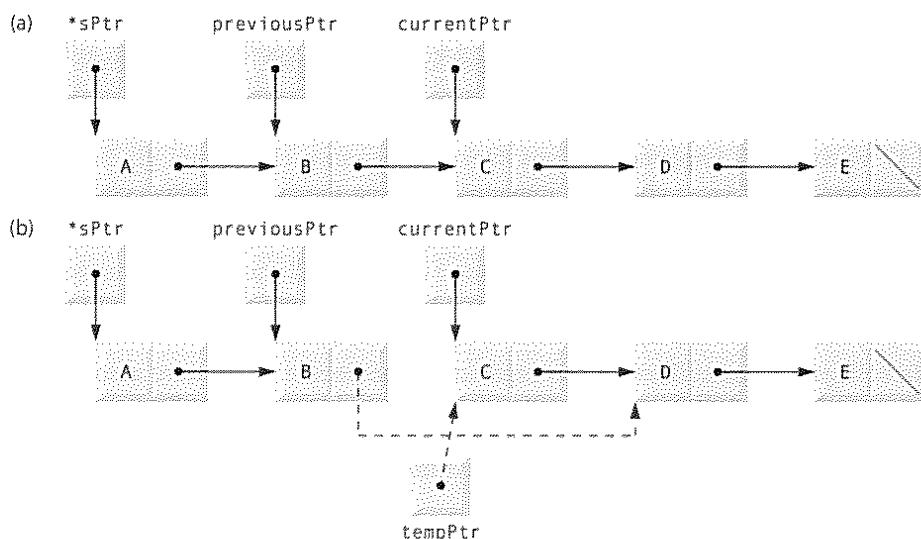
La funzione `delete` effettua i seguenti passi.

- Se il carattere da cancellare corrisponde al carattere nel primo nodo (riga 115), dobbiamo eliminare il primo nodo. Assegnate `*sPtr` a `tempPtr`, che useremo per liberare la memoria del nodo. Assegnate `(*sPtr)->nextPtr` a `*sPtr`, in modo che `startPtr` in `main` ora punti a quello che prima era il *secondo* nodo della lista. Chiamate `free` per deallocare la memoria puntata da `tempPtr`. Restituite il carattere che è stato cancellato.
- Altrimenti, inizializzate `previousPtr` con `*sPtr` e inizializzate `currentPtr` con `(*sPtr)->nextPtr` (righe 122-123) per avanzare fino al secondo nodo.
- Localizzate il carattere da cancellare. Finché `currentPtr` non è `NULL` e il valore da cancellare non è uguale a `currentPtr->data` (riga 126), assegnate `currentPtr` a `previousPtr` (riga 127) e assegnate `currentPtr->nextPtr` a `currentPtr` (riga 128) per avanzare fino al nodo successivo della lista.
- Se `currentPtr` non è `NULL` (riga 132), il carattere è contenuto nella lista. Assegnate `currentPtr` a `tempPtr` (riga 133), che useremo per deallocare il nodo. Assegnate `currentPtr->nextPtr` a `previousPtr->nextPtr` (riga 134) per collegare il nodo che precede il nodo rimosso con il nodo che lo segue. Liberate il nodo a cui punta `tempPtr` (riga 135) e restituite il carattere cancellato (riga 136).

Se non viene restituito nulla, la riga 140 restituisce il carattere nullo ('\0') per indicare che il carattere da cancellare non è stato trovato nella lista.

Illustrare un'eliminazione

Il seguente diagramma illustra l'eliminazione del nodo contenente 'C' da una lista collegata. La parte (a) della figura mostra la lista collegata prima dell'eliminazione. La parte (b) mostra la riassegnazione del collegamento. Il puntatore tempPtr viene usato per liberare la memoria allocata al nodo che memorizza 'C'. Notate che nelle righe 118 e 135 liberiamo tempPtr. Precedentemente, abbiamo raccomandato di impostare un puntatore liberato a NULL. Qui non lo facciamo perché tempPtr è una variabile automatica locale e la funzione restituisce il controllo immediatamente dopo la liberazione della memoria.



12.4.3 Funzioni isEmpty e printList

La funzione `isEmpty` (righe 144-146) è una **funzione predicato**: essa non altera in alcun modo la lista. Piuttosto, `isEmpty` determina se la lista è vuota, cioè se il puntatore al primo nodo è `NULL`. Se la lista è vuota, `isEmpty` restituisce 1; altrimenti, restituisce 0.

```

143 // restituisci 1 se la lista e' vuota, altrimenti 0
144 int isEmpty(ListNodePtr sPtr) {
145     return sPtr == NULL;
146 }
147
148 // stampa la lista
149 void printList(ListNodePtr currentPtr) {
150     // se la lista e' vuota
151     if (isEmpty(currentPtr)) {
152         puts("List is empty.\n");
153     }
154     else {
155         puts("The list is:");
156
157         // finche' non si raggiunge la fine della lista
158         while (currentPtr != NULL) {
159             printf("%c --> ", currentPtr->data);

```

```

160         currentPtr = currentPtr->nextPtr;
161     }
162
163     puts("NULL\n");
164 }
165 }
```

La funzione `printList` (righe 149-165) stampa una lista. Il parametro `currentPtr` della funzione riceve un puntatore al primo nodo della lista. La funzione in primo luogo determina se la lista è vuota (righe 151-153) e, se lo è, stampa "List is empty." e termina. Altrimenti, le righe 155-163 stampano i dati nella lista. Finché `currentPtr` non è `NULL`, la riga 159 stampa il valore in `currentPtr->data`, e la riga 160 assegna `currentPtr->nextPtr` a `currentPtr` per avanzare al nodo successivo. Se il collegamento nell'ultimo nodo della lista non è `NULL`, l'algoritmo di stampa cercherà di stampare oltre la fine della lista, e si verificherà un errore logico. Questo algoritmo di stampa è identico per le liste collegate, le pile e le code.



Esercizi di ricorsione basati su liste

L'Esercizio 12.17 vi chiede di implementare una funzione ricorsiva che stampi gli elementi in una lista in ordine inverso. L'Esercizio 12.18 vi chiede di implementare una funzione ricorsiva che cerchi un particolare elemento in una lista collegata.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è *falsa*?

- a) Una lista collegata è una collezione con organizzazione lineare di strutture autoreferenziali, chiamate nodi, connesse da puntatori di collegamento (link), da cui il termine lista "collegata".
- b) A una lista collegata si accede mediante un puntatore al primo nodo della lista.
- c) In una lista collegata si accede ai nodi successivi mediante il puntatore di collegamento memorizzato in ogni nodo.
- d) Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

2. (*Vero/Falso*) Una lista collegata è appropriata quando il numero di elementi da memorizzare nella struttura di dati non è prevedibile.

Risposta: *Vero*.

3. (*Vero/Falso*) Gli elementi di una lista collegata sono registrati nella memoria in modo contiguo. Questo permette l'accesso immediato a ciascun elemento, perché l'indirizzo dell'elemento può essere calcolato direttamente in base alla sua posizione rispetto all'inizio della lista collegata. Gli array non offrono un tale accesso immediato ai loro elementi.

Risposta: *Falso*. In realtà, è vero l'opposto. Gli elementi di un array sono registrati nella memoria in modo contiguo e supportano l'accesso immediato per posizione. Le liste collegate non offrono un tale accesso immediato ai loro elementi.

4. (*Completare*) Passare in una funzione l'indirizzo di un puntatore al primo nodo di una lista collegata crea un puntatore a un puntatore. Questo viene spesso chiamato _____ doppia.

Risposta: indirezione.

12.5 Pile

Una pila può essere implementata come una versione vincolata di una lista collegata. È possibile aggiungere nuovi nodi ed eliminare quelli esistenti solo in cima. Per questa ragione, una pila è detta struttura di dati **last-in, first-out (LIFO)**: il primo elemento a uscire è l'ultimo entrato. A una pila si fa riferimento tramite un puntatore all'elemento in cima a essa. Il membro `link` nell'ultimo nodo della pila è impostato a `NULL` per indicare il fondo della pila. Non terminare una pila con `NULL` può causare errori in fase di esecuzione.



Il seguente diagramma illustra una pila con diversi nodi: `stackPtr` punta all'elemento in cima alla pila. Le pile e le liste collegate sono rappresentate in modo identico in queste figure; la differenza tra loro è che gli inserimenti e le cancellazioni possono avvenire ovunque in una lista collegata, ma solo in cima alla pila.



Principali operazioni per le pile

Le principali funzioni usate per manipolare una pila sono `push` e `pop`. La funzione `push` crea un nuovo nodo e lo colloca in cima alla pila. La funzione `pop` rimuove un nodo dalla cima della pila, libera la memoria che era stata allocata per il nodo rimosso e restituisce il valore del nodo.

Implementare una pila

Il programma della Figura 12.2 implementa una semplice pila di interi. Il programma permette di effettuare un `push` di un valore nella pila (funzione `push`), effettuare un `pop` di un valore dalla pila (funzione `pop`) e terminare l'esecuzione. Le righe 7-10 definiscono `struct stackNode`, che useremo per rappresentare i nodi della pila. Come mostrato nella Figura 12.1, usiamo `typedef` (righe 12-13) per rendere più leggibile il codice. Inizialmente, `stackPtr` (riga 23) è impostato a `NULL` per indicare una pila vuota. Gran parte della logica di questa applicazione è simile alla Figura 12.1, quindi qui ci concentriamo sulle differenze.

```

1 // fig12_02.c
2 // Un semplice programma che usa una pila
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 // struttura autoreferenziale
7 struct stackNode {
8     int data; // il campo dati e' un int
9     struct stackNode *nextPtr; // puntatore a stackNode
10 };
11
12 typedef struct stackNode StackNode; // sinonimo per struct stackNode
13 typedef StackNode *StackNodePtr; // sinonimo per StackNode*
14
15 // prototipi
16 void push(StackNodePtr *topPtr, int info);
17 int pop(StackNodePtr *topPtr);
18 int isEmpty(StackNodePtr topPtr);
19 void printStack(StackNodePtr currentPtr);
20 void instructions(void);
21
22 int main(void) {
23     StackNodePtr stackPtr = NULL; // punta alla cima della pila
24     int value = 0; // valore int inserito dall'utente
25
26     instructions(); // stampa il menu
27     printf("%s", "? ");
28     int choice = 0; // scelta da menu da parte dell'utente
29     scanf("%d", &choice);
30
31     // finche' l'utente non inserisce 3

```

```
32     while (choice != 3) {
33         switch (choice) {
34             case 1: // effettua un push di un valore nella pila
35                 printf("%s", "Enter an integer: ");
36                 scanf("%d", &value);
37                 push(&stackPtr, value);
38                 printStack(stackPtr);
39                 break;
40             case 2: // effettua un pop di un valore dalla pila
41                 // se la pila non e' vuota
42                 if (!isEmpty(stackPtr)) {
43                     printf("The popped value is %d.\n", pop(&stackPtr));
44                 }
45                 printStack(stackPtr);
46                 break;
47             default:
48                 puts("Invalid choice.\n");
49                 instructions();
50                 break;
51         }
52     }
53
54     printf("%s", "? ");
55     scanf("%d", &choice);
56 }
57
58 puts("End of run.");
59 }
60
61 // stampa le istruzioni del programma per l'utente
62 void instructions(void) {
63     puts("Enter choice:\n"
64         "1 to push a value on the stack\n"
65         "2 to pop a value off the stack\n"
66         "3 to end program");
67 }
68
69 // inserisci un nodo in cima alla pila
70 void push(StackNodePtr *topPtr, int info) {
71     StackNodePtr newPtr = malloc(sizeof(StackNode));
72
73     // inserisci il nodo in cima alla pila
74     if (newPtr != NULL) {
75         newPtr->data = info;
76         newPtr->nextPtr = *topPtr;
77         *topPtr = newPtr;
78     }
79     else { // non c'e' spazio disponibile
80         printf("%d not inserted. No memory available.\n", info);
81     }
82 }
83
84 // rimuovi un nodo dalla cima della pila
85 int pop(StackNodePtr *topPtr) {
```

```

86     StackNodePtr tempPtr = *topPtr;
87     int popValue = (*topPtr)->data;
88     *topPtr = (*topPtr)->nextPtr;
89     free(tempPtr);
90     return popValue;
91 }
92
93 // stampa la pila
94 void printStack(StackNodePtr currentPtr) {
95     if (currentPtr == NULL) { // se la pila e' vuota
96         puts("The stack is empty.\n");
97     }
98     else {
99         puts("The stack is:");
100
101        while (currentPtr != NULL) { // finche' non si raggiunge la fine della pila
102            printf("%d --> ", currentPtr->data);
103            currentPtr = currentPtr->nextPtr;
104        }
105
106        puts("NULL\n");
107    }
108 }
109
110 // restituisci 1 se la pila e' vuota, altrimenti 0
111 int isEmpty(StackNodePtr topPtr) {
112     return topPtr == NULL;
113 }

```

```

Enter choice:
1 to push a value on the stack
2 to pop a value off the stack
3 to end program
? 1

Enter an integer: 5
The stack is:
5 --> NULL

? 1
Enter an integer: 6
The stack is:
6 --> 5 --> NULL

? 1
Enter an integer: 4
The stack is:
4 --> 6 --> 5 --> NULL

? 2
The popped value is 4.
The stack is:
6 --> 5 --> NULL

```

```

? 2
The popped value is 6.
The stack is: 5 --> NULL
?
? 2
The popped value is 5.
The stack is empty.
?
? 2
The stack is empty.
?
? 4
Invalid choice.
Enter choice:
1 to push a value on the stack
2 to pop a value off the stack
3 to end program
? 3
End of run.

```

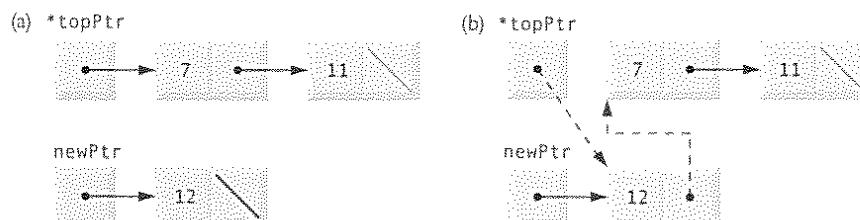
Figura 12.2 Un semplice programma che usa una pila.

12.5.1 Funzione push

La funzione push (righe 70-82) inserisce un nuovo nodo in cima alla pila usando i seguenti passi.

- Chiamate `malloc` per creare un nuovo nodo, quindi assegnate l'indirizzo della memoria allocata a `newPtr` (riga 71).
- Assegnate a `newPtr->data` il valore da mettere nella pila (riga 75) e assegnate `*topPtr` (il puntatore alla cima della pila) a `newPtr->nextPtr` (riga 76). Il membro `link` del nuovo nodo in cima ora punta al precedente nodo in cima.
- Assegnate `newPtr` a `*topPtr` (riga 77): ora `stackPtr` in `main` punta alla nuova cima della pila.

Il seguente diagramma illustra un'operazione di push. La parte (a) della figura mostra la pila e il nuovo nodo prima che l'operazione di push inserisca il nuovo nodo in cima alla pila; `*topPtr` rappresenta `stackPtr` in `main`. Le frecce tratteggiate nella parte (b) illustrano i passi 2 e 3 dell'analisi precedente, nei quali viene inserito in cima alla pila il nodo contenente 12.

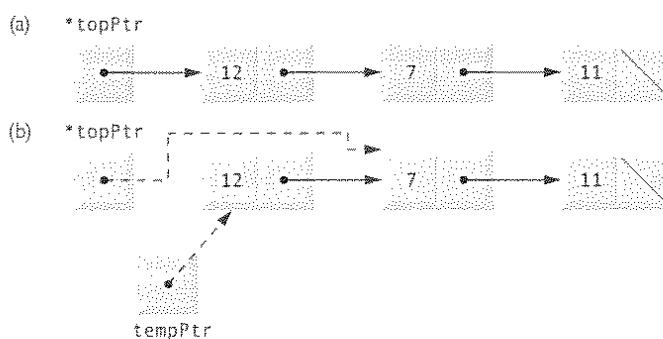


12.5.2 Funzione pop

La funzione pop (righe 85-91) estrae un nodo dalla cima della pila. La funzione `main` determina se la pila è vuota prima di chiamare `pop`. L'operazione `pop` è costituita da cinque passi.

1. Assegnate `*topPtr` a `tempPtr` (riga 86), che sarà usato per liberare la memoria del nodo.
2. Assegnate `(*topPtr)->data` a `popValue` (riga 87) per recuperare il valore del nodo in cima in modo da poterlo restituire.
3. Assegnate `(*topPtr)->nextPtr` a `*topPtr` (riga 88) in modo che `stackPtr` in `main` ora punti a quello che precedentemente era il secondo elemento della pila (o `NULL` se non c'erano altri elementi).
4. Liberare la memoria puntata da `tempPtr` (riga 89).
5. Restituite `popValue` alla funzione chiamante (riga 90).

Il seguente diagramma illustra un'operazione di pop. La parte (a) mostra la pila prima dell'eliminazione del nodo contenente 12; `*topPtr` rappresenta `stackPtr` in `main`. La parte (b) mostra `tempPtr` che punta al nodo che deve essere sottoposto a un'operazione di pop e `*topPtr` che punta al nuovo nodo in cima. Poi possiamo liberare la memoria puntata da `tempPtr`.



12.5.3 Applicazioni delle pile

Le pile hanno molte applicazioni interessanti. Per esempio, ogni volta che viene effettuata una chiamata a una funzione, la funzione chiamata deve sapere come ritornare alla sua funzione chiamante, pertanto l'indirizzo di ritorno è inserito con un push in una pila (Paragrafo 5.7). Se si ha una serie di chiamate a funzioni, gli indirizzi successivi di ritorno sono inseriti nella pila nell'ordine last-in, first-out, in modo che ogni funzione possa ritornare alla sua funzione chiamante. Le pile supportano le chiamate di funzioni ricorsive nello stesso modo delle chiamate convenzionali non ricorsive.

Le pile mantengono lo spazio creato per le variabili automatiche locali in ogni invocazione di una funzione. Quando la funzione ritorna alla sua funzione chiamante, lo spazio per le variabili automatiche di quella funzione viene eliminato con un pop dalla pila e queste variabili non sono più note al programma. Le pile possono essere usate dai compilatori anche nel processo di valutazione delle espressioni e di generazione di codice in linguaggio macchina. Gli esercizi esplorano diverse applicazioni delle pile.

✓ Autovalutazione

1. **(Scelta multipla)** Quale delle seguenti affermazioni è falsa?
 - a) Una pila deve essere implementata come una versione vincolata di una lista collegata.
 - b) I nuovi nodi sono aggiunti e rimossi da un pila solo in cima.
 - c) Una pila è una struttura di dati last-in, first-out (LIFO).
 - d) A una pila si fa riferimento tramite un puntatore all'elemento in cima alla pila stessa.
- Risposta:** (a) è falsa. In realtà, una pila può essere implementata come una versione vincolata di una lista collegata, ma non è necessario che lo sia.
2. **(Scelta multipla)** Quale delle seguenti affermazioni a), b) o c) è falsa?
 - a) Quando una funzione viene chiamata, deve sapere come ritornare alla sua funzione chiamante, pertanto l'indirizzo di ritorno della funzione chiamata viene inserito con un push nella pila.

- b) In una serie di chiamate a funzioni, gli indirizzi successivi di ritorno sono inseriti nella pila nell'ordine last-in, first-out, in modo che ogni funzione possa ritornare alla sua funzione chiamante.
- c) Le pile supportano le chiamate di funzioni ricorsive nello stesso modo delle chiamate convenzionali non ricorsive.
- d) Tutte le affermazioni precedenti sono vere.

Risposta: (a) è falsa. In realtà, l'indirizzo di ritorno della *funzione chiamante* viene inserito con un push nella pila.

12.6 Code

Una **coda** è simile alla fila davanti alla cassa in un supermercato:

- la prima persona in fila è servita per prima;
- gli altri clienti si aggiungono alla fine della coda e aspettano di essere serviti.

I nodi della coda si estraggono solo dalla **testa della coda** e si inseriscono solo alla **fine della coda**. Per questa ragione, una coda è chiamata struttura di dati **first-in, first-out (FIFO)**: il primo elemento a entrare è il primo a uscire. Le operazioni di inserimento e di estrazione sono note, rispettivamente, come enqueue e dequeue.

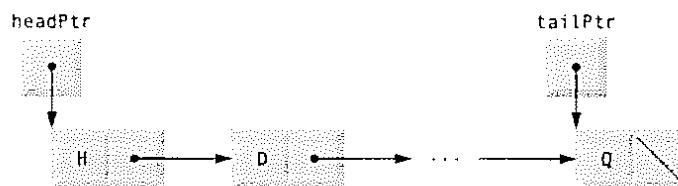
Applicazioni delle code

Le code hanno molte applicazioni nei sistemi di elaborazione.

- Nei computer che hanno soltanto un singolo processore è possibile servire solo un utente alla volta. Le richieste degli altri utenti sono poste in una coda. Ogni richiesta avanza gradualmente verso la testa della coda man mano che le richieste di servizio vengono soddisfatte. L'elemento in testa alla coda è il prossimo a ricevere il servizio.
- Analogamente, per i sistemi multicore odierni, potrebbero esserci più utenti che processori, quindi i programmi che non sono ancora in esecuzione vengono collocati in una coda finché non diverrà disponibile un processore che al momento è occupato. Nell'Appendice C parleremo di multithreading. Quando il lavoro di un programma viene suddiviso in più thread in grado di funzionare in parallelo, potrebbero esserci più thread che processori, quindi i thread che al momento non sono in esecuzione devono rimanere in attesa in una coda.
- Le code supportano anche lo spooling di stampa. Un ufficio può avere un'unica stampante e molti utenti possono generare allo stesso tempo più documenti da stampare. Quando la stampante è occupata, i documenti aggiuntivi vengono inseriti nella memoria o nella memoria secondaria, proprio come il filo per cucire che viene avvolto attorno a un rocchetto (spool) finché non è necessario. I documenti attendono in coda fino a quando la stampante non diventa disponibile.
- Anche i pacchetti di informazioni che viaggiano su reti di computer, come Internet, aspettano in coda. Ogni volta che arriva un pacchetto al nodo di una rete, deve essere instradato al nodo successivo sulla rete lungo il percorso verso la sua destinazione finale. Il nodo che effettua l'instradamento (router) inoltra un pacchetto alla volta, pertanto gli ulteriori pacchetti sono messi in coda in attesa di essere instradati.

Illustrare una coda

Il seguente diagramma illustra una coda con diversi nodi. Notate i puntatori separati alla testa e alla fine della coda. Come con le liste e le pile, se non si imposta a NULL il collegamento nell'ultimo nodo della coda possono verificarsi errori logici.



Implementare una coda

Il programma della Figura 12.3 esegue operazioni su una coda. Il programma fornisce diverse opzioni per inserire un nodo nella coda (funzione `enqueue`), eliminare un nodo dalla coda (funzione `dequeue`) e terminare il programma. Le righe 7-10 definiscono `struct queueNode`, che useremo per rappresentare i nodi della coda. Anche in questo caso, usiamo `typedef` (righe 12-13) per rendere il codice più leggibile. Gran parte della logica di questa applicazione è simile ai nostri esempi di liste e pile, quindi qui ci soffermeremo sulle differenze. Inizialmente, `headPtr` e `tailPtr` (righe 23-24) sono entrambi `NULL` per indicare una coda vuota.

```

1 // fig12_03.c
2 // Gestione di una coda
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 // struttura autoreferenziale
7 struct queueNode {
8     char data; // il campo dati e' un char
9     struct queueNode *nextPtr; // puntatore a queueNode
10 };
11
12 typedef struct queueNode QueueNode;
13 typedef QueueNode *QueueNodePtr;
14
15 // prototipi di funzioni
16 void printQueue(QueueNodePtr currentPtr);
17 int isEmpty(QueueNodePtr headPtr);
18 void enqueue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value);
19 char dequeue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr);
20 void instructions(void);
21
22 int main(void) {
23     QueueNodePtr headPtr = NULL; // inizializza headPtr
24     QueueNodePtr tailPtr = NULL; // inizializza tailPtr
25     char item = '\0'; // char inserito dall'utente
26
27     instructions(); // stampa il menu
28     printf("%s", "? ");
29     int choice = 0; // scelta da menu da parte dell'utente
30     scanf("%d", &choice);
31
32     // finche' l'utente non inserisce 3
33     while (choice != 3) {
34         switch(choice) {
35             case 1: // metti in coda un valore
36                 printf("%s", "Enter a character: ");
37                 scanf("\n%c", &item);
38                 enqueue(&headPtr, &tailPtr, item);
39                 printQueue(headPtr);
40                 break;
41             case 2: // estrai dalla coda un valore
42                 // se la coda non e' vuota
43                 if (!isEmpty(headPtr)) {
44                     item = dequeue(&headPtr, &tailPtr);
45                     printf("%c has been dequeued.\n", item);

```

```
46         }
47
48         printQueue(headPtr);
49         break;
50     default:
51         puts("Invalid choice.\n");
52         instructions();
53         break;
54     }
55
56     printf("%s", "? ");
57     scanf("%d", &choice);
58 }
59
60 puts("End of run.");
61 }
62
63 // stampa le istruzioni del programma per l'utente
64 void instructions(void) {
65     printf ("Enter your choice:\n"
66             "    1 to add an item to the queue\n"
67             "    2 to remove an item from the queue\n"
68             "    3 to end\n");
69 }
70
71 // inserisci un nodo in fondo alla coda
72 void enqueue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value) {
73     QueueNodePtr newPtr = malloc(sizeof(QueueNode));
74
75     if (newPtr != NULL) { // c'e' spazio disponibile?
76         newPtr->data = value;
77         newPtr->nextPtr = NULL;
78
79         // se la coda e' vuota, inserisci il nodo in testa
80         if (isEmpty(*headPtr)) {
81             *headPtr = newPtr;
82         }
83         else {
84             (*tailPtr)->nextPtr = newPtr;
85         }
86
87         *tailPtr = newPtr;
88     }
89     else {
90         printf("%c not inserted. No memory available.\n", value);
91     }
92 }
93
94 // rimuovi un nodo dalla testa della coda
95 char dequeue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr) {
96     char value = (*headPtr)->data;
97     QueueNodePtr tempPtr = *headPtr;
98     *headPtr = (*headPtr)->nextPtr;
```

```

99
100 // se la coda e' vuota
101 if (*headPtr == NULL) {
102     *tailPtr = NULL;
103 }
104
105 free(tempPtr);
106 return value;
107 }
108
109 // restituisci 1 se la coda e' vuota, altrimenti 0
110 int isEmpty(QueueNodePtr headPtr) {
111     return headPtr == NULL;
112 }
113
114 // stampa la coda
115 void printQueue(QueueNodePtr currentPtr) {
116     if (currentPtr == NULL) { // se la coda e' vuota
117         puts("Queue is empty.\n");
118     }
119     else {
120         puts("The queue is:");
121
122         while (currentPtr != NULL) { // finche' non si raggiunge la fine della coda
123             printf("%c --> ", currentPtr->data);
124             currentPtr = currentPtr->nextPtr;
125         }
126
127         puts("NULL\n");
128     }
129 }

```

```

Enter your choice:
1 to add an item to the queue
2 to remove an item from the queue
3 to end
? 1 to add item.
Enter a character: A
The queue is:
A --> NULL
? 1 to add item.
Enter a character: B
The queue is:
A --> B --> NULL
? 1 to add item.
Enter a character: C
The queue is:
A --> B --> C --> NULL

```

```

? 2
A has been dequeued.
The queue is:.....B --> C --> NULL
? 2
B has been dequeued.
The queue is:.....C --> NULL
? 2
C has been dequeued.
Queue is empty.
? 2
Queue is empty.
? 4
Invalid choice.
Enter your choice:
1 to add an item to the queue
2 to remove an item from the queue
3 to end
? 3
End of run.

```

Figura 12.3 Gestione di una coda.

12.6.1 Funzione enqueue

La funzione enqueue (righe 72-92) riceve tre argomenti:

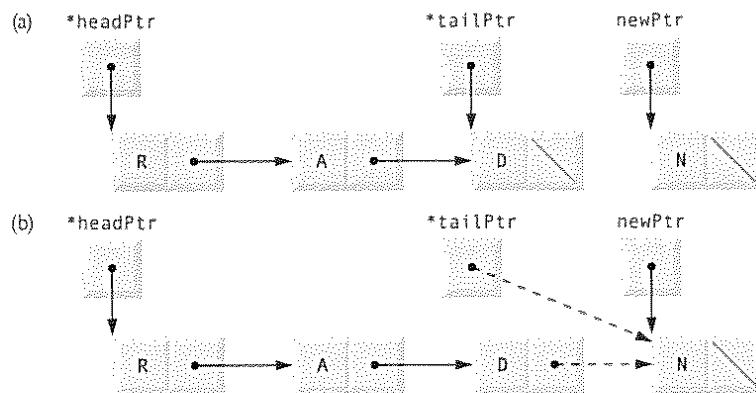
- l'indirizzo di headPtr (il puntatore alla testa della coda);
- l'indirizzo di tailPtr (il puntatore alla fine della coda);
- il valore da inserire.

La funzione effettua i seguenti passi.

1. La riga 73 chiama malloc per creare un nuovo nodo e assegna la posizione allocata in memoria a newPtr.
2. Se la memoria è stata allocata correttamente, le righe 76 e 77 assegnano il valore da inserire a newPtr->data, e assegnano NULL a newPtr->nextPtr.
3. Se la coda è vuota (riga 80), la riga 81 assegna newPtr a *headPtr poiché il nuovo nodo è sia la testa che la fine della coda; altrimenti, la riga 84 assegna newPtr a (*tailPtr)->nextPtr poiché il nuovo nodo è il nuovo nodo della fine della coda.
4. La riga 87 assegna newPtr a *tailPtr per aggiornare il puntatore della fine della coda al nuovo nodo della fine della coda.

Illustrare un'operazione di enqueue

Il seguente diagramma illustra un'operazione di enqueue. La parte (a) mostra `headPtr` e `tailPtr` di `main` e il nuovo nodo di enqueue prima dell'operazione. Le frecce tratteggiate nella parte (b) illustrano i passi 3 e 4 della precedente analisi, in cui si aggiunge il nuovo nodo alla fine di una coda non vuota.



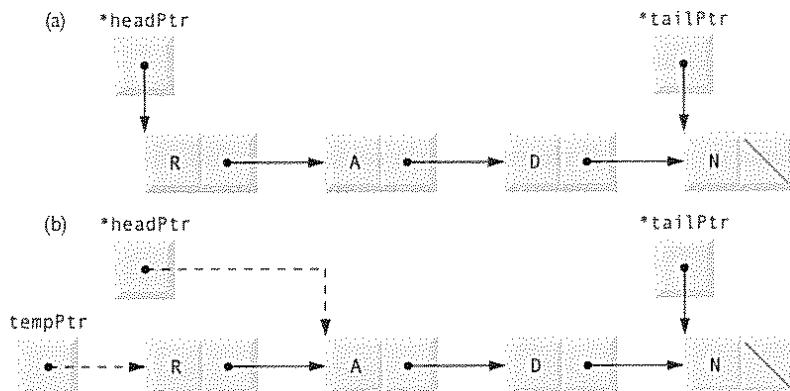
12.6.2 Funzione dequeue

La funzione `dequeue` (righe 95-107) riceve come argomenti gli indirizzi dei puntatori alla testa e alla fine della coda e rimuove il primo nodo della coda. L'operazione di `dequeue` effettua i seguenti passi.

1. La riga 96 assegna `(*headPtr)->data` a `value` per salvare i dati che vengono rimossi dalla coda.
2. La riga 97 assegna `*headPtr` a `tempPtr`, che sarà usato per liberare la memoria.
3. La riga 98 assegna `(*headPtr)->nextPtr` a `*headPtr` in modo che il puntatore alla testa della coda in `main` ora punti al nuovo nodo in testa alla coda.
4. La riga 101 controlla se `*headPtr` è `NULL`. Se lo è, la riga 102 assegna `NULL` a `*tailPtr` poiché la coda ora è vuota.
5. La riga 105 libera la memoria puntata da `tempPtr`.
6. La riga 106 restituisce `value` alla funzione chiamante.

Illustrare un'operazione di dequeue

Il seguente diagramma illustra la funzione `dequeue`. La parte (a) mostra la coda prima dell'operazione di `dequeue`: `*headPtr` e `*tailPtr` in `dequeue` sono usati per modificare `headPtr` e `tailPtr` di `main`. La parte (b) mostra `tempPtr` che punta al nodo rimosso dalla coda, e `headPtr` di `main` aggiornato per puntare al nuovo primo nodo della coda.



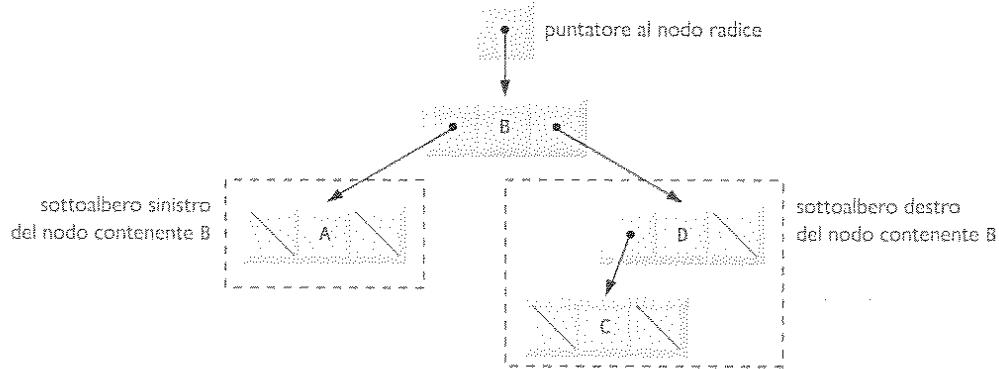
✓ Autovalutazione

1. **(Scelta multipla)** Quale delle seguenti affermazioni sulle code è *falsa*?
 - a) Una coda è simile alla fila davanti alla cassa in un supermercato: la prima persona in fila è servita per prima; gli altri clienti si aggiungono alla fila solo alla fine della coda e aspettano di essere serviti.
 - b) I nodi della coda si estraggono solo dall'inizio della coda (head) e si inseriscono solo alla fine della coda (tail).
 - c) Una coda è una struttura di dati first-in, last-out (FILO).
 - d) Le operazioni di inserimento e di estrazione sono note, rispettivamente, come enqueue e dequeue.

Risposta: (c) è *falsa*. In realtà, una coda è una struttura di dati first-in, first-out (FIFO).
 2. **(Completare)** Per i sistemi _____ odierni, potrebbero esserci più programmi in esecuzione che processori, quindi i programmi che non sono ancora in esecuzione vengono collocati in una coda finché non diverrà disponibile un processore che al momento è occupato.
- Risposta:** multicore.

12.7 Alberi

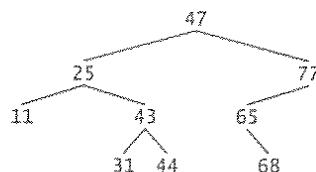
Fin qui abbiamo presentato strutture di dati lineari: liste collegate, pile e code. Un **albero** è una struttura di dati bidimensionale non lineare con speciali proprietà. I nodi degli alberi contengono due o più collegamenti. Questa sezione analizza gli **alberi binari**, alberi i cui nodi contengono due collegamenti, come nel seguente diagramma:



Nessuno, uno o entrambi i collegamenti di ciascun nodo possono essere NULL. Il **nodo radice** è il primo nodo in un albero. Ogni collegamento nel nodo radice si riferisce a un **child**. Il **figlio a sinistra** è il primo nodo nel **sottoalbero sinistro** e il **figlio a destra** è il primo nel **sottoalbero destro**. I figli di uno stesso nodo sono chiamati **fratelli**. Un nodo senza figli è chiamato **nodo foglia**. Se i collegamenti di un nodo foglia non vengono impostati a NULL possono verificarsi errori in fase di esecuzione. Gli informatici normalmente disegnano alberi con il nodo radice in cima, esattamente l'opposto degli alberi in natura.

Albero di ricerca binaria

Questo paragrafo presenta un **albero di ricerca binaria** contenente valori unici, la cui caratteristica è che i valori di ogni sottoalbero *sinistro* sono minori del valore nel **nodo padre** corrispondente e i valori in ogni sottoalbero *destro* sono maggiori del valore nel nodo padre corrispondente. La figura seguente illustra un albero di ricerca binaria con nove valori:



La forma dell'albero di ricerca binaria per un insieme di dati può variare a seconda dell'ordine in cui sono inseriti i suoi valori.

Implementare un albero di ricerca binaria

Il programma della Figura 12.4 crea un albero di ricerca binaria e ne attraversa i nodi, ossia visita ogni nodo dell'albero per "fare qualcosa" con i valori in esso contenuti, come visualizzarli. L'albero viene attraversato in tre modi: **in ordine**, **in pre-ordine** e **in post-ordine**. Il programma genera 10 numeri a caso e inserisce ognuno nell'albero, ma i valori duplicati vengono eliminati. Le righe 9-13 definiscono `struct treeNode`, che useremo per rappresentare i nodi dell'albero. Anche in questo caso, usiamo `typedef` (righe 15-16) per rendere il codice più leggibile.

```

1 // fig12_04.c
2 // Creazione e attraversamento di un albero binario
3 // in pre-ordine, in ordine e in post-ordine
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <time.h>
7
8 // struttura autoreferenziale
9 struct treeNode {
10     struct treeNode *leftPtr; // puntatore al sottoalbero sinistro
11     int data; // valore del nodo
12     struct treeNode *rightPtr; // puntatore al sottoalbero destro
13 };
14
15 typedef struct treeNode TreeNode; // sinonimo per struct treeNode
16 typedef TreeNode *TreeNodePtr; // sinonimo per TreeNode*
17
18 // prototipi
19 void insertNode(TreeNodePtr *treePtr, int value);
20 void inOrder(TreeNodePtr treePtr);
21 void preOrder(TreeNodePtr treePtr);
22 void postOrder(TreeNodePtr treePtr);
23
24 int main(void) {
25     TreeNodePtr rootPtr = NULL; // albero inizialmente vuoto
26
27     srand(time(NULL));
28     puts("The numbers being placed in the tree are:");
29
30     // inserisci nell'albero valori a caso tra 0 e 14
31     for (int i = 1; i <= 10; ++i) {
32         int item = rand() % 15;
33         printf("%3d", item);
34         insertNode(&rootPtr, item);
35     }
36
37     // attraversa l'albero in pre-ordine
38     puts("\n\nThe preOrder traversal is:");
39     preOrder(rootPtr);
40
41     // attraversa l'albero in ordine
42     puts("\n\nThe inOrder traversal is:");

```

```
43     inOrder(rootPtr);
44
45     // attraversa l'albero in post-ordine
46     puts("\n\nThe postOrder traversal is:");
47     postOrder(rootPtr);
48 }
49
50 // inserisci un nodo nell'albero
51 void insertNode(TreeNodePtr *treePtr, int value) {
52     if (*treePtr == NULL) { // se l'albero e' vuoto
53         *treePtr = malloc(sizeof(TreeNode));
54
55         if (*treePtr != NULL) { // se la memoria e' stata allocata, assegna i dati
56             (*treePtr)->data = value;
57             (*treePtr)->leftPtr = NULL;
58             (*treePtr)->rightPtr = NULL;
59         }
60     else {
61         printf("%d not inserted. No memory available.\n", value);
62     }
63 }
64 else { // l'albero non e' vuoto
65     if (value < (*treePtr)->data) { // il valore va nel sottoalbero sinistro
66         insertNode(&(*treePtr)->leftPtr), value);
67     }
68     else if (value > (*treePtr)->data) { // il valore va nel sottoalbero destro
69         insertNode(&(*treePtr)->rightPtr), value);
70     }
71     else { // i valori dei dati duplicati vengono ignorati
72         printf("%s", "dup");
73     }
74 }
75 }
76
77 // inizia l'attraversamento in ordine dell'albero
78 void inOrder(TreeNodePtr treePtr) {
79     // se l'albero non e' vuoto, allora traversalo
80     if (treePtr != NULL) {
81         inOrder(treePtr->leftPtr);
82         printf("%3d", treePtr->data);
83         inOrder(treePtr->rightPtr);
84     }
85 }
86
87 // inizia l'attraversamento in pre-ordine dell'albero
88 void preOrder(TreeNodePtr treePtr) {
89     // se l'albero non e' vuoto, allora traversalo
90     if (treePtr != NULL) {
91         printf("%3d", treePtr->data);
92         preOrder(treePtr->leftPtr);
93         preOrder(treePtr->rightPtr);
94     }
}
```

```

95 }
96 // inizia l'attraversamento in post-ordine dell'albero
97 void postOrder(TreeNodePtr treePtr) {
98     // se l'albero non e' vuoto, allora attraversalo
99     if (treePtr != NULL) {
100         postOrder(treePtr->leftPtr);
101         postOrder(treePtr->rightPtr);
102         printf("%3d", treePtr->data);
103     }
104 }
105 }
```

The numbers being placed in the tree are:

6 7 4 12 7dup 2 2dup 5 7dup 11

The preOrder traversal is:

6 4 2 5 7 12 11

The inOrder traversal is:

2 4 5 6 7 11 12

The postOrder traversal is:

2 5 4 11 12 7 6

Figura 12.4 Creazione e attraversamento di un albero binario.

12.7.1 Funzione insertNode

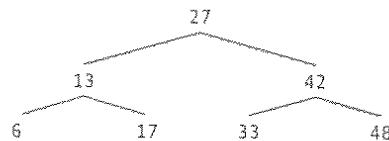
Le funzioni usate nella Figura 12.4 per creare un albero di ricerca binaria e attraversarlo sono ricorsive. La funzione `insertNode` (righe 51-75) riceve come argomenti l'indirizzo del puntatore al nodo radice dell'albero e un intero da inserire. Ogni nuovo nodo in un albero di ricerca binaria inizialmente viene inserito come nodo foglia. Di seguito sono descritti i passi per inserire un nodo.

- Se `*treePtr` è `NULL` (riga 52), la riga 53 chiama `malloc` per creare un nuovo nodo foglia e assegna la memoria allocata a `*treePtr`. La riga 56 assegna a `(*treePtr)->data` l'intero da memorizzare. Le righe 57 e 58 assegnano `NULL` a `(*treePtr)->leftPtr` e `(*treePtr)->rightPtr`. Poi il controllo ritorna alla funzione chiamante, o `main` o una chiamata precedente a `insertNode`.
- Se `*treePtr` non è `NULL` e il valore da inserire è minore di `(*treePtr)->data`, la riga 66 chiama ricorsivamente `insertNode` con l'indirizzo di `(*treePtr)->leftPtr` per inserire il nuovo nodo nel sottoalbero sinistro del nodo puntato da `treePtr`.
- Se il valore da inserire è maggiore di `(*treePtr)->data`, la riga 69 chiama ricorsivamente `insertNode` con l'indirizzo di `(*treePtr)->rightPtr` per inserire il nodo nel sottoalbero destro del nodo puntato da `treePtr`.

I passi ricorsivi continuano finché `insertNode` non trova un puntatore `NULL`, poi il *passo 1* inserisce il nuovo nodo come nodo foglia.

12.7.2 Attraversamenti: funzioni inOrder, preOrder e postOrder

Le funzioni `inOrder` (righe 78-85), `preOrder` (righe 88-95) e `postOrder` (righe 98-105) ricevono ognuna un puntatore al nodo radice di un albero e attraversano (ovvero visitano) l'albero. Descriveremo gli attraversamenti e mostreremo il risultato dell'applicazione di ciascuno di essi per l'albero illustrato qui di seguito.



Attraversamento in ordine

Ecco i passi per un attraversamento in ordine (funzione `inOrder`).

1. Attraversate il sottoalbero *sinistro* `inOrder` (riga 81).
2. Elaborate il valore nel nodo (riga 82).
3. Attraversate il sottoalbero *destro* `inOrder` (riga 83).

Con questa procedura viene elaborato il valore di ciascun nodo dopo l'elaborazione dei valori nel sottoalbero sinistro. L'attraversamento in ordine del precedente albero è:

6 13 17 27 33 42 48

L'attraversamento in ordine di un albero di ricerca binaria elabora i nodi in ordine crescente. Il processo di creazione di un albero di ricerca binaria in realtà mette in ordine i dati, e per questo è chiamato **ordinamento con albero binario**.

Attraversamento in pre-ordine

Ecco i passi per un attraversamento in pre-ordine (funzione `preOrder`).

1. Elaborate il valore nel nodo (riga 91).
2. Attraversate il sottoalbero *sinistro* in pre-ordine (riga 92).
3. Attraversate il sottoalbero *destro* in pre-ordine (riga 93).

Questo attraversamento elabora il valore di ogni nodo quando il nodo viene visitato. Dopo l'elaborazione del valore, un attraversamento in pre-ordine elabora i valori del sottoalbero sinistro, poi i valori del sottoalbero destro. L'attraversamento in pre-ordine del precedente albero è:

27 13 6 17 42 33 48

Attraversamento in post-ordine

Ecco i passi per un attraversamento in post-ordine (funzione `postOrder`).

1. Attraversate il sottoalbero *sinistro* in post-ordine (riga 101).
2. Attraversate il sottoalbero *destro* in post-ordine (riga 102).
3. Elaborate il valore nel nodo (riga 103).

Questo attraversamento elabora il valore di ogni nodo dopo l'elaborazione dei valori dei figli del nodo in entrambi i sottoalberi sinistro e destro. L'attraversamento in post-ordine del precedente albero è:

6 17 13 33 48 42 27

12.7.3 Eliminazione dei duplicati

L'albero di ricerca binaria facilita l'**eliminazione dei duplicati**. Quando inserite valori per creare l'albero, un valore duplicato seguirà su ogni confronto le stesse decisioni "andare a sinistra" o "andare a destra" del valore originario. Pertanto, il duplicato verrà alla fine confrontato con un nodo nell'albero contenente lo stesso valore. A questo punto, il valore duplicato può essere ignorato.

12.7.4 Ricerca in un albero binario

La ricerca in un albero binario di un valore che corrisponde a un valore chiave è anche veloce. Se l'albero è ben compattato, ogni livello contiene all'incirca il doppio degli elementi del livello precedente. Un albero di ricerca binaria con n elementi avrà quindi un massimo di $\log_2 n$ livelli e di conseguenza si dovranno effettuare un *massimo* di $\log_2 n$ confronti per trovare una corrispondenza o per determinare che non ne esistano. Quando si effettua una ricerca in un albero di ricerca binaria ben compattato con 1.000 elementi, sono necessari non più di 10 confronti, perché $2^{10} > 1.000$. Quando si effettua una ricerca in un albero di ricerca binaria (ben compattato) con 1.000.000 di elementi, sono necessari non più di 20 confronti, perché $2^{20} > 1.000.000$.

12.7.5 Altre operazioni su alberi binari

Negli esercizi vengono presentati algoritmi per diverse altre operazioni su alberi binari, come la stampa di un albero binario in un formato ad albero bidimensionale e l'esecuzione di un attraversamento per livelli successivi di un albero binario. L'attraversamento per livelli successivi visita i nodi dell'albero riga per riga iniziando dal livello del nodo radice. In ogni livello dell'albero i nodi vengono visitati da sinistra a destra. Gli altri esercizi sugli alberi binari riguardano la possibilità per un albero di ricerca binaria di contenere valori duplicati, la creazione di un albero di stringhe e la determinazione del numero di livelli di un albero binario.

✓ Autovalutazione

1. *(Vero/Falso)* Liste collegate, pile, code e alberi sono strutture di dati lineari.

Risposta: *Falso*. In realtà, gli alberi sono strutture di dati bidimensionali non lineari.

2. *(Completare)* Tre modi comuni per attraversare un albero di ricerca binaria sono _____, in pre-ordine e in post-ordine.

Risposta: in ordine.

3. *(Completare)* Il processo di creazione di un albero di ricerca binaria in realtà _____ i dati.

Risposta: ordina.

4. *(Vero/Falso)* La forma dell'albero di ricerca binaria per un insieme di dati è indipendente dall'ordine in cui i valori sono inseriti nell'albero.

Risposta: *Falso*. In realtà, la forma dell'albero di ricerca binaria per un insieme di dati varia a seconda dell'ordine in cui i valori sono inseriti nell'albero.

5. *(Vero/Falso)* Un nodo può essere inserito in un albero di ricerca binaria solo come nodo radice.

Risposta: *Falso*. In realtà, un nodo può essere inserito in un albero di ricerca binaria solo come nodo *foglia*.

6. *(Completare)* Un albero di ricerca binaria facilita l'_____. Mentre vengono inseriti i valori per creare un albero, i valori identici seguiranno su ogni confronto le stesse decisioni di "andare a destra" o "andare a sinistra" del valore originario. Un valore identico verrà alla fine confrontato con un nodo nell'albero contenente lo stesso valore.

Risposta: eliminazione dei duplicati.

7. *(Vero/Falso)* Un albero binario ben compattato con n elementi avrà un massimo di circa $\log_2 n$ livelli. Effettuare una ricerca in un tale albero richiede un massimo di circa $\log_2 n$ confronti sia per trovare una corrispondenza sia per determinare che non ne esistano. Di conseguenza, quando si effettua una ricerca in un albero di ricerca binaria (ben compattato) con 1.000.000.000 di elementi sono necessari non più di 20 confronti.

Risposta: *Falso*. In realtà, sono necessari non più di 30 confronti.

12.8 Programmazione sicura in C

Il Capitolo 8 del *SEI CERT C Coding Standard* è dedicato alle raccomandazioni e alle regole riguardanti la gestione della memoria (molte si riferiscono all'uso dei puntatori e alla gestione dinamica della memoria trattati in questo capitolo). Per maggiori informazioni, visitate il sito <https://wiki.sei.cmu.edu/>.

- MEM01-C/MEM30-C: i puntatori dovrebbero essere sempre inizializzati con NULL o l'indirizzo di un elemento valido in memoria. Quando usate `free` per rilasciare dinamicamente la memoria allocata, al puntatore passato a `free` non viene assegnato un nuovo valore, per cui esso continua a puntare all'indirizzo della memoria che era stata allocata in modo dinamico. L'uso di un tale puntatore "sospeso" può portare ad arresti anomali del programma e a vulnerabilità della sicurezza. Quando liberate la memoria allocata dinamicamente, dovete immediatamente assegnare al puntatore o NULL o un indirizzo valido. Scegliamo di non fare ciò per variabili puntatore locali che immediatamente escono dal campo d'azione dopo una chiamata a `free`.
- MEM01-C: quando tentate di usare `free` per rilasciare memoria dinamica che è stata già rilasciata, si verifica un comportamento indefinito, noto come "vulnerabilità del doppio `free`". Per assicurarvi che non tentiate di rilasciare più di una volta la stessa memoria, impostate immediatamente il puntatore a NULL dopo la chiamata a `free`. Tentare di liberare un puntatore NULL non ha alcun effetto.
- ERR33-C: la maggior parte delle funzioni della Libreria Standard restituisce valori che consentono di determinare se le funzioni hanno eseguito le loro operazioni correttamente. La funzione `malloc`, per esempio, restituisce NULL se non è in grado di allocare la memoria necessaria. Dovete sempre assicurarvi che `malloc` non restituisca NULL prima di tentare di usare il puntatore che memorizza il valore di ritorno di `malloc`.

✓ Autovalutazione

1. (*Vero/Falso*) I puntatori devono essere sempre inizializzati a NULL.

Risposta: *Falso*. In realtà, i puntatori devono essere sempre inizializzati a NULL o all'indirizzo di un elemento valido in memoria.

2. (*Vero/Falso*) Per evitare di rilasciare due volte la stessa memoria e causare così un comportamento indefinito, impostate il puntatore a NULL dopo la chiamata a `free`.

Risposta: *Vero*. Liberare un puntatore NULL non causa un comportamento indefinito.

3. (*Completare*) Se la funzione `malloc` non è in grado di allocare la memoria necessaria, restituisce _____. Verificate sempre se questo è il valore restituito prima di usare il puntatore che memorizza il valore di ritorno di `malloc`.

Risposta: NULL.

12.9 Riepilogo

Paragrafo 12.1 Introduzione

- Le strutture di dati dinamiche crescono e si riducono in fase di esecuzione.
- Le liste collegate sono collezioni di dati "allineati in una riga". In una lista collegata gli inserimenti e le cancellazioni vengono effettuati ovunque.
- Con le pile, gli inserimenti e le cancellazioni si effettuano solo in cima.
- Le code rappresentano file di attesa. Potete inserire elementi solo alla fine della coda (indicata con tail) ed eliminarne solo all'inizio della coda (indicato con head).
- Gli alberi binari facilitano la ricerca e l'ordinamento ad alta velocità di dati, l'eliminazione efficiente di dati duplicati, la rappresentazione di directory di file system e la compilazione di espressioni nel linguaggio macchina.

Paragrafo 12.2 Strutture autoreferenziali

- Una struttura autoreferenziale contiene un membro puntatore che punta a una struttura dello stesso tipo.
- Le strutture autoreferenziali possono essere collegate insieme per formare liste, code, pile e alberi.
- Un puntatore `NULL` indica normalmente la fine di una struttura di dati.

Paragrafo 12.3 Gestione dinamica della memoria

- La creazione e il mantenimento di strutture di dati dinamiche richiedono la gestione dinamica di memoria.
- Le funzioni `malloc` e `free` e l'operatore `sizeof` sono essenziali per l'allocazione dinamica di memoria.
- La funzione `malloc` riceve il numero di byte da allocare e restituisce un puntatore `void *` alla memoria allocata. Un puntatore `void *` può essere assegnato a una variabile di un qualsiasi tipo di puntatore.
- Se non c'è memoria disponibile, `malloc` restituisce `NULL`.
- La funzione `free` rilascia la memoria, in modo da poterla riallocare in futuro.
- Il C fornisce anche le funzioni `calloc` e `realloc` per creare e modificare array dinamici.

Paragrafo 12.4 Liste collegate

- Una lista collegata è una collezione con organizzazione lineare di strutture autoreferenziali, chiamate nodi, connesse da collegamenti tramite puntatori.
- Una lista collegata è accessibile mediante un puntatore al primo nodo. I nodi successivi sono accessibili mediante il membro puntatore di collegamento (link) memorizzato in ogni nodo.
- Per convenzione, il puntatore di collegamento nell'ultimo nodo di una lista è impostato a `NULL` per segnare la fine della lista.
- In una lista collegata i dati sono memorizzati in modo dinamico. Ogni nodo viene creato quando è necessario.
- Un nodo può contenere dati di ogni tipo compresi altri oggetti `struct`.
- Le liste collegate sono dinamiche, perciò la lunghezza di una lista può aumentare o diminuire quando è necessario.
- I nodi delle liste collegate non sono normalmente contenuti in memoria in modo contiguo. Dal punto di vista logico, tuttavia, i nodi di una lista collegata appaiono contigui.

Paragrafo 12.5 Pile

- Una pila si può implementare come una versione vincolata di una lista collegata. I nuovi nodi possono essere aggiunti e rimossi da una pila solo dalla cima.
- Una pila viene indicata come una struttura di dati last-in, first-out (LIFO).
- Le principali funzioni usate per manipolare una pila sono `push` e `pop`. La funzione `push` crea un nuovo nodo e lo pone in cima alla pila. La funzione `pop` estrae un nodo dalla cima della pila, libera la memoria che era allocata per il nodo estratto e restituisce il valore estratto.
- Ogni volta che viene effettuata una chiamata a una funzione, la funzione chiamata deve sapere come tornare alla sua funzione chiamante, pertanto l'indirizzo di ritorno è inserito in cima a una pila. Se si verifica una serie di chiamate a funzioni, i valori di ritorno successivi sono inseriti nella pila in ordine last-in, first-out, in modo che ogni funzione possa tornare alla sua funzione chiamante. Le pile supportano le chiamate di funzioni ricorsive alla stessa maniera delle chiamate non ricorsive convenzionali.
- Le pile sono usate dai compilatori nel processo di valutazione di espressioni e di generazione del codice in linguaggio macchina.

Paragrafo 12.6 Code

- I nodi di una coda vengono rimossi solo dalla testa della coda e inseriti solo in fondo alla coda. Una coda viene pertanto indicata come una struttura di dati first-in, first-out (FIFO).
- Le operazioni di inserimento e di rimozione di elementi per una coda sono note come enqueue e dequeue.

Paragrafo 12.7 Alberi

- Un albero è una struttura di dati non lineare e bidimensionale. I nodi di un albero contengono due o più collegamenti.
- Gli alberi binari sono alberi i cui nodi contengono tutti due collegamenti.
- Il nodo radice è il primo nodo in un albero. Ogni collegamento nel nodo radice di un albero binario si riferisce a un figlio. Il figlio sinistro è il primo nodo nel sottoalbero sinistro e il figlio destro è il primo nodo nel sottoalbero destro. I figli di un nodo sono chiamati fratelli.
- Un nodo senza figli è chiamato nodo foglia.
- Un albero di ricerca binaria (senza valori duplicati) ha la caratteristica che i valori in ogni sottoalbero sinistro sono minori del valore nel nodo padre corrispondente e i valori in ogni sottoalbero destro sono maggiori del valore nel nodo padre corrispondente.
- Un nodo può essere inserito in un albero di ricerca binaria solo come nodo foglia.
- I passi per un attraversamento in ordine sono: attraversa il sottoalbero sinistro in ordine, elabora il valore nel nodo, quindi attraversa il sottoalbero destro in ordine. Il valore in ogni nodo non è elaborato finché non sono stati elaborati i valori nel suo sottoalbero sinistro.
- L'attraversamento in ordine di un albero di ricerca binaria elabora i valori dei nodi in ordine crescente. Il processo di creazione di un albero di ricerca binaria in realtà ordina i dati. Pertanto questo processo viene chiamato ordinamento con albero binario.
- I passi per un attraversamento in pre-ordine sono: elabora il valore nel nodo, attraversa il sottoalbero sinistro in pre-ordine, quindi attraversa il sottoalbero destro in pre-ordine. Dopo che viene elaborato il valore in un dato nodo, vengono elaborati prima i valori nel sottoalbero sinistro, poi quelli nel sottoalbero destro.
- I passi per un attraversamento in post-ordine sono: attraversa il sottoalbero sinistro in post-ordine, attraversa il sottoalbero destro in post-ordine, quindi elabora il valore nel nodo. Il valore in ogni nodo non viene elaborato finché non sono stati elaborati i valori dei suoi figli.
- Un albero di ricerca binaria facilita l'eliminazione di duplicati. Quando si crea un albero, un tentativo di inserire un valore duplicato sarà riconosciuto perché un duplicato seguirà su ogni confronto le stesse decisioni "andare a sinistra" o "andare a destra" del valore originario. Pertanto, il duplicato verrà alla fine confrontato con un nodo nell'albero contenente lo stesso valore. A questo punto il valore duplicato potrà essere semplicemente eliminato.
- La ricerca in un albero binario di un valore corrispondente a un valore chiave è veloce. Se l'albero è ben compattato, ogni livello contiene circa il doppio degli elementi del livello precedente. Pertanto un albero di ricerca binaria con n elementi avrebbe un massimo di $\log_2 n$ livelli e si dovrebbe quindi effettuare un massimo di $\log_2 n$ confronti sia per trovare una corrispondenza sia per determinare che non ne esiste alcuna. Ciò significa che, quando si effettua una ricerca in un albero di ricerca binaria con 1.000 elementi ben compattato, servono non più di 10 confronti, perché $2^{10} > 1.000$. Quando si effettua una ricerca in un albero di ricerca binaria con 1.000.000 di elementi ben compattato, servono non più di 20 confronti poiché $2^{20} > 1.000.000$.

Esercizi di autovalutazione

- 12.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
- Una struttura auto-_____ viene usata per costruire strutture dinamiche di dati.
 - La funzione _____ viene usata per allocare in modo dinamico la memoria.

- c) Una _____ è una versione specializzata di una lista collegata nella quale i nodi possono essere inseriti e cancellati solo dall'inizio della lista.
- d) Le funzioni che esaminano una lista collegata ma non la modificano sono chiamate _____.
- e) Una coda viene detta una struttura di dati _____.
- f) Il puntatore al nodo successivo in una lista collegata è chiamato _____.
- g) La funzione _____ viene usata per rilasciare la memoria allocata in modo dinamico.
- h) Una _____ è una versione specializzata di una lista collegata nella quale i nodi possono essere inseriti solo alla fine della lista e rimossi solo dall'inizio di essa.
- i) Un _____ è una struttura di dati non lineare e bidimensionale contenente nodi con due o più collegamenti.
- j) Una pila è chiamata struttura di dati _____ perché l'ultimo nodo inserito è il primo a essere rimosso.
- k) I nodi di un albero _____ contengono due membri link.
- l) Il primo nodo di un albero è il nodo _____.
- m) Ogni collegamento nel nodo di un albero punta a un _____ o a un _____ di quel nodo.
- n) Un nodo di un albero che non ha figli è chiamato nodo _____.
- o) I tre algoritmi di attraversamento (trattati in questo capitolo) per un albero binario sono detti _____, _____ e _____.

12.2 (*Discussione*) Quali sono le differenze tra una lista collegata e una pila?

12.3 (*Discussione*) Quali sono le differenze tra una pila e una coda?

12.4 Scrivete un'istruzione o un insieme di istruzioni per eseguire ognuna delle seguenti operazioni. Supponete che tutte le manipolazioni avvengano nella funzione `main` (dunque non occorrono indirizzi di variabili puntatore) e presupponete le seguenti definizioni:

```
struct gradeNode {
    char lastName[20];
    double grade;
    struct gradeNode *nextPtr;
};

typedef struct gradeNode GradeNode;
typedef GradeNode *GradeNodePtr;
```

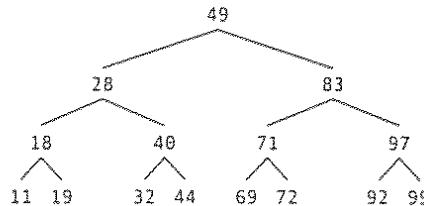
- a) Create un puntatore all'inizio della lista con il nome `startPtr`. La lista è vuota.
- b) Create un nuovo nodo di tipo `GradeNode` puntato dal puntatore `newPtr` di tipo `GradeNodePtr`. Assegnate la stringa "Jones" al membro `lastName` e il valore 91.5 al membro `grade` (usate `strcpy`). Scrivete tutte le dichiarazioni e le istruzioni necessarie.
- c) Supponete che la lista puntata da `startPtr` sia composta attualmente da 2 nodi: uno contenente "Jones" e uno contenente "Smith". I nodi sono in ordine alfabetico. Scrivete le istruzioni necessarie per inserire i nodi contenenti i seguenti dati per `lastName` e `grade`, assicurandovi di inserire i nodi in ordine:

"Adams"	85.0
"Thompson"	73.5
"Pritchard"	66.5

Usate i puntatori `previousPtr`, `currentPtr` e `newPtr` per effettuare gli inserimenti. Prima di ogni inserimento, stabilite a cosa puntano `previousPtr` e `currentPtr`. Supponete che `newPtr` punti sempre al nuovo nodo e che a questo siano già stati assegnati i dati.

- d) Scrivete un ciclo `while` che stampi i dati di ogni nodo della lista. Usate il puntatore `currentPtr` per spostarvi lungo la lista.
- e) Scrivete un ciclo `while` che cancelli tutti i nodi nella lista e liberi la memoria associata a ogni nodo. Usate il puntatore `currentPtr` e il puntatore `tempPtr`, rispettivamente per muovervi lungo la lista e per liberare memoria.

12.5 (Attraversamenti di un albero di ricerca binaria) Indicate le sequenze di valori relative agli attraversamenti in ordine, pre-ordine e post-ordine del seguente albero di ricerca binaria.



Risposte agli esercizi di autovalutazione

12.1 a) referenziale. b) malloc. c) pila. d) predicati. e) FIFO. f) link. g) free. h) coda. i) albero. j) LIFO. k) binario. l) radice. m) figlio, sottoalbero. n) foglia. o) in ordine, pre-ordine, post-ordine.

12.2 È possibile inserire e rimuovere un nodo ovunque in una lista collegata. Tuttavia, i nodi in una pila possono essere inseriti e rimossi solo dalla cima della pila.

12.3 Una coda ha puntatori sia alla testa che alla fine di essa, in modo che i nodi possano essere inseriti alla fine e rimossi dalla testa. Una pila ha un solo puntatore alla cima, dove si effettuano sia l'inserimento che la rimozione dei nodi.

12.4

- GradeNodePtr startPtr = NULL;
- GradeNodePtr newPtr;

```

newPtr = malloc(sizeof(GradeNode));
strcpy(newPtr->lastName, "Jones");
newPtr->grade = 91.5;
newPtr->nextPtr = NULL;

```
- Per inserire "Adams";

```

previousPtr is NULL, currentPtr punta al primo elemento nella lista.
newPtr->nextPtr = currentPtr;
startPtr = newPtr;

```
- Per inserire "Thompson";

```

previousPtr punta all'ultimo elemento nella lista (contenente "Smith")
currentPtr is NULL.
newPtr->nextPtr = currentPtr;
previousPtr->nextPtr = newPtr;

```
- Per inserire "Pritchard";

```

previousPtr punta al nodo contenente "Jones"
currentPtr punta al nodo contenente "Smith"
newPtr->nextPtr = currentPtr;
previousPtr->nextPtr = newPtr;

```
- currentPtr = startPtr;

```

while (currentPtr != NULL) {
    printf("Lastname = %s\nGrade = %.2f\n",
           currentPtr->lastName, currentPtr->grade);
    currentPtr = currentPtr->nextPtr;
}

```
- currentPtr = startPtr;

```

while (currentPtr != NULL) {
    tempPtr = currentPtr;
    currentPtr = currentPtr->nextPtr;
    free(tempPtr);
}
startPtr = NULL;

```

- 12.5 L'attraversamento *in ordine* è: 11 18 19 28 32 40 44 49 69 71 72 83 92 97 99
 L'attraversamento in *pre-ordine* è: 49 28 18 11 19 40 32 44 83 71 69 72 97 92 99
 L'attraversamento in *post-ordine* è: 11 19 18 32 44 40 28 69 72 71 92 99 97 83 49

Esercizi

12.6 (*Concatenamento di liste*) Scrivete un programma che concatensi due liste collegate di caratteri. Il programma deve includere la funzione *concatenate* che riceve come argomenti i puntatori a entrambe le liste e concatena la seconda lista alla prima.

12.7 (*Fusione di liste ordinate*) Scrivete un programma in grado di fondere due liste ordinate di interi in una singola lista ordinata. La funzione *merge* deve ricevere i puntatori al primo nodo di ognuna delle liste da fondere e restituire un puntatore al primo nodo della lista fusa.

12.8 (*Inserimento in una lista ordinata*) Scrivete un programma che inserisca a caso 25 interi da 0 a 100 in ordine in una lista collegata. Il programma deve calcolare la somma degli elementi e la loro media in virgola mobile.

12.9 (*Creare una lista collegata, quindi invertirne gli elementi*) Scrivete un programma che crei una lista collegata di 10 caratteri e poi crei una copia della lista in ordine inverso.

12.10 (*Invertire le parole di una frase*) Scrivete un programma che riceva in ingresso una riga di testo e usi una pila per stampare la riga in ordine inverso.

12.11 (*Test di palindromi*) Scrivete un programma che usi una pila per determinare se una stringa è un palindromo (cioè la stringa si legge allo stesso modo in avanti e all'indietro). Il programma deve ignorare spazi e punteggiatura.

12.12 (*Simulazione di un supermercato*) Scrivete un programma che simuli una fila alla cassa di un supermercato. La fila è una coda. I clienti arrivano a intervalli interi casuali da 1 a 4 minuti. Inoltre, ogni cliente viene servito a intervalli interi casuali da 1 a 4 minuti. Ovviamente, le frequenze devono essere equilibrate. Se la frequenza media di arrivo è più grande della frequenza media del servizio, la coda crescerà all'infinito. Persino con frequenze equilibrate la casualità può provocare ancora lunghe file. Eseguite la simulazione del supermercato per una giornata di 12 ore (720 minuti) usando il seguente algoritmo.

1. Scegliete un intero a caso tra 1 e 4 per determinare il minuto in cui arriva il primo cliente.
2. Al momento dell'arrivo del primo cliente:
 - determinate il tempo del servizio per il cliente (un intero casuale da 1 a 4);
 - iniziate a servire il cliente;
 - calcolate il momento di arrivo del cliente successivo (un intero casuale da 1 a 4 aggiunto al tempo corrente).
3. Per ogni minuto del giorno:
 - Se arriva il cliente successivo,
 - mostratelo;
 - mettete in coda il cliente;
 - calcolate il momento di arrivo del cliente successivo.
 - Se il servizio per l'ultimo cliente servito è stato completato,
 - mostratelo;
 - estraete dalla coda il prossimo cliente da servire;
 - determinate il momento del completamento del servizio per il cliente (un intero casuale da 1 a 4 aggiunto al tempo corrente).

Ora fate eseguire la vostra simulazione per 720 minuti e rispondete a ognuna delle seguenti domande.

- a) Qual è il numero massimo di clienti nella coda a una qualunque ora?
- b) Qual è l'attesa più lunga di un cliente?
- c) Cosa succede se l'intervalllo degli arrivi viene cambiato da 1 a 4 minuti a un intervallo da 1 a 3 minuti?

12.13 (Consentire duplicati in un albero binario) Modificate il programma della Figura 12.4 per permettere che l'albero binario contenga valori duplicati.

12.14 (Albero di ricerca binaria di stringhe) Scrivete un programma basato su quello della Figura 12.4 che riceva in ingresso una riga di testo, suddivida la frase in parole separate, inserisca le parole in un albero di ricerca binaria e stampi i risultati degli attraversamenti dell'albero in ordine, pre-ordine e post-ordine.

Leggete la riga di testo e memorizzatela in un array. Usate `strtok` per suddividere in token il testo. Quando viene rilevato un token, create un nuovo nodo per l'albero, assegnate il puntatore restituito da `strtok` al membro `string` del nuovo nodo e inserite il nodo nell'albero.

12.15 (Eliminazione dei duplicati) Abbiamo visto che l'eliminazione dei duplicati è semplice quando si crea un albero di ricerca binaria. Descrivete come eseguireste l'eliminazione dei duplicati usando soltanto un array unidimensionale. Confrontate l'esecuzione dell'eliminazione dei duplicati basata su un array con l'esecuzione dell'eliminazione dei duplicati basata sull'albero di ricerca binaria.

12.16 (Profondità di un albero binario) Scrivete una funzione `depth` che riceva un albero binario e determini quanti livelli ha.

12.17 (Stampare ricorsivamente una lista all'indietro) Scrivete una funzione `printListBackward` che stampi in maniera ricorsiva gli elementi in una lista in ordine inverso. Usate la vostra funzione in un programma di test che crea una lista ordinata di interi e stampa la lista in ordine inverso.

12.18 (Effettuare ricorsivamente una ricerca in una lista) Scrivete una funzione `searchList` che cerchi ricorsivamente un valore specificato in una lista collegata. La funzione deve restituire un puntatore al valore se questo viene trovato, altrimenti deve restituire `NULL`. Usate la vostra funzione in un programma di test che crea una lista di interi. Il programma deve poi richiedere all'utente un valore da cercare nella lista.

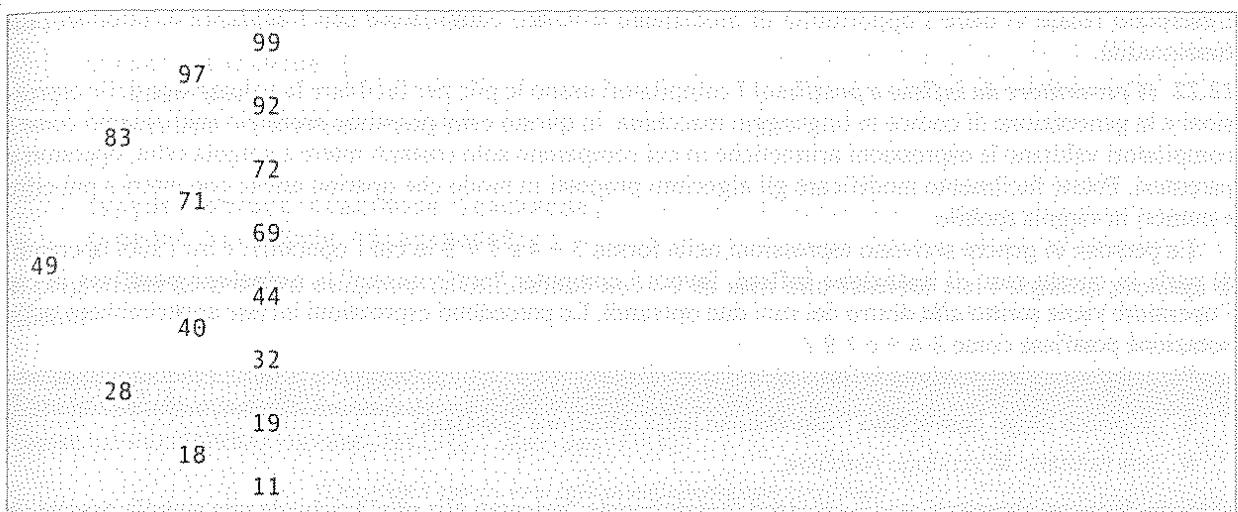
12.19 (Ricerca in un albero binario) Scrivete la funzione `binaryTreeSearch` che cerca di localizzare un valore specificato in un albero di ricerca binaria. La funzione deve ricevere come argomenti un puntatore al nodo radice dell'albero binario e una chiave di ricerca. Se viene trovato un nodo contenente la chiave di ricerca, la funzione deve restituire un puntatore a quel nodo; altrimenti, la funzione deve restituire un puntatore `NULL`.

12.20 (Attraversamento di un albero binario per livelli successivi) Il programma della Figura 12.4 illustra tre metodi ricorsivi di attraversamento di un albero binario: in ordine, in pre-ordine e in post-ordine. Questo esercizio presenta l'**attraversamento per livelli successivi** di un albero binario. Questo attraversamento elabora i valori dei nodi livello per livello da sinistra a destra partendo dal livello del nodo radice. L'attraversamento per livelli successivi non è un algoritmo ricorsivo. Esso usa la struttura di dati a coda per elaborare i nodi nell'ordine corretto. L'algoritmo è il seguente.

1. Inserite il nodo radice nella coda.
2. Finché vi sono nodi nella coda,
 - estraete il nodo successivo dalla coda.
 - Stampate il valore del nodo.
 - Se il puntatore al figlio sinistro del nodo non è `NULL`,
 - inserite nella coda il nodo relativo al figlio sinistro.
 - Se il puntatore al figlio destro del nodo non è `NULL`,
 - inserite nella coda il nodo relativo al figlio destro.

Scrivete la funzione `levelOrder` per compiere un attraversamento per livelli successivi di un albero binario. La funzione deve ricevere come argomento un puntatore al nodo radice dell'albero binario. Modificate il programma della Figura 12.4 per usare questa funzione. Confrontate l'output di questa funzione con l'output degli altri attraversamenti per controllare che abbia funzionato correttamente. In questo programma dovrete modificare e incorporare le funzioni di elaborazione di code della Figura 12.3.

12.21 (Stampare alberi) Scrivete una funzione ricorsiva `outputTree` per stampare sullo schermo un albero binario. La funzione deve stampare l'albero riga per riga, con la cima dell'albero alla sinistra dello schermo e la parte inferiore dell'albero verso la destra dello schermo. Ogni riga viene stampata verticalmente. Per esempio, l'albero binario dell'Esercizio 12.5 è stampato come segue:



Notate che il nodo foglia più a destra appare in cima all'output nella colonna più a destra e che il nodo radice appare alla sinistra dell'output. Ogni colonna di output inizia cinque spazi a destra della colonna precedente. La funzione `outputTree` deve ricevere come argomenti un puntatore al nodo radice dell'albero e un intero `totalSpaces` che indica il numero di spazi che deve precedere il valore da stampare. Questa variabile deve iniziare da zero, in modo che il nodo radice sia stampato alla sinistra dello schermo. La funzione usa un attraversamento in ordine modificato per stampare l'albero. L'algoritmo è il seguente:

Finché il puntatore al nodo corrente non è `NULL`,
 chiamate ricorsivamente `outputTree` con il sottoalbero destro del nodo corrente e
`totalSpaces + 5`.
 Usate un'istruzione `for` per contare da 1 a `totalSpaces` e stampare gli spazi.
 Stampate il valore nel nodo corrente.
 Chiamate ricorsivamente `outputTree` con il sottoalbero sinistro del nodo corrente e
`totalSpaces + 5`.

Paragrafo speciale: caso pratico sul software dei sistemi – costruire il proprio compilatore

Nell'Esercizio 7.28 abbiamo introdotto il linguaggio creato da noi SML (Simpletron Machine Language). Nell'Esercizio 7.29 avete usato una simulazione per creare il computer Simpletron (una *macchina virtuale*) che esegue programmi scritti in SML. In questo paragrafo impegnativo costruirete un compilatore che converte in SML i programmi scritti in **Simple**, un linguaggio di programmazione inventato, conciso e di alto livello. In questo paragrafo vengono collegate tutte insieme le varie parti del processo di programmazione. Dovrete:

- scrivere diversi programmi nel linguaggio di alto livello Simple;
- compilare i programmi con il compilatore da voi costruito, generando codice in linguaggio macchina SML in un file;
- caricare il codice in linguaggio macchina SML dal suddetto file nella memoria del Simpletron;
- eseguire i programmi in linguaggio macchina SML sulla macchina virtuale Simpletron che avete costruito nell'Esercizio 7.29.

Questo paragrafo è composto da sei esercizi. I primi due sono relativi ad alcune principali tecnologie informatiche necessarie per implementare il vostro compilatore. Il terzo introduce il linguaggio di alto livello Simple con alcuni esempi completi di codice e vi richiede di scrivere diversi programmi in Simple. Il quarto vi guida nella costruzione del vostro compilatore. Il quinto introduce il tema fondamentale dell'ottimizzazione del compilatore: modificherete il vostro compilatore per ridurre il numero di istruzioni in SML da esso generate, rendendo così i vostri programmi in SML più efficienti in termini di memoria e di velocità di esecuzione.

L'esercizio finale vi offre l'opportunità di modificare il vostro compilatore con l'aggiunta di ulteriori utili funzionalità.

12.22 (Convertitore da infisso a postfisso) I compilatori usano le pile per facilitare la valutazione delle espressioni e la generazione di codice in linguaggio macchina. In questo e nel prossimo esercizio analizziamo come i compilatori valutano le espressioni aritmetiche in cui compaiono solo costanti intere a singola cifra, operatori e parentesi. Potete facilmente modificare gli algoritmi proposti in modo che operino anche con interi a più cifre e numeri in virgola mobile.

Le persone in genere scrivono espressioni nella forma $3 + 4$ e $7 / 9$ in cui l'operatore è *tra* i suoi operandi. Si parla in questo caso di **notazione infissa**. Invece i computer "preferiscono" la **notazione postfissa** in cui l'operatore viene scritto *alla destra* dei suoi due operandi. Le precedenti espressioni infisse apparirebbero nella notazione postfissa come $3\ 4 +$ e $7\ 9 /$.

Per valutare un'espressione infissa, alcuni compilatori

- in primo luogo convertono l'espressione nella notazione postfissa;
- poi valutano la versione postfissa.

Ognuno di questi **algoritmi stack-oriented** (letteralmente "orientati alla pila") richiede una singola passata da sinistra a destra dell'espressione. In questo esercizio, implemerete l'**algoritmo di conversione da infisso a postfisso**. Nell'esercizio successivo, implemerete l'**algoritmo per la valutazione dell'espressione postfissa**.

Scrivete un programma che converta una corretta espressione aritmetica infissa con interi a singola cifra come

$(6 + 2) * 5 - 8 / 4$

in un'espressione postfissa. La versione postfissa dell'espressione infissa precedente è

$6\ 2 + 5 * 8\ 4 / -$

Osservate che l'espressione postfissa non contiene parentesi. Il programma deve leggere l'espressione memorizzata nell'array di caratteri `infix` e usare le funzioni per le pile implementate in questo capitolo come supporto alla creazione dell'espressione postfissa nell'array di caratteri `postfix`. L'algoritmo per la creazione di un'espressione postfissa è il seguente.

1. Inserite con un push una parentesi sinistra '(' nella pila.
2. Aggiungete una parentesi destra ')' alla fine di `infix`.
3. Finché la pila non è vuota, leggete `infix` da sinistra a destra ed effettuate le seguenti operazioni.
 - Se il carattere corrente di `infix` è una cifra, copiatelo nel successivo elemento di `postfix`.
 - Se il carattere corrente di `infix` è una parentesi sinistra, inseritelo con un push nella pila.
 - Se il carattere corrente di `infix` è un operatore,
 - estraete (pop) gli operatori (se ve ne sono) dalla cima della pila finché questi hanno precedenza uguale o maggiore rispetto all'operatore corrente e inserite gli operatori estratti in `postfix`;
 - inserite il carattere corrente in `infix` nella pila.
 - Se il carattere corrente di `infix` è una parentesi destra,
 - estraete (pop) gli operatori dalla cima della pila e inseriteli in `postfix` finché non compare una parentesi sinistra in cima alla pila;
 - estraete (ed eliminate) la parentesi sinistra dalla pila.

In un'espressione sono permesse le seguenti operazioni aritmetiche:

- + addizione
- sottrazione
- * moltiplicazione
- / divisione

La pila deve essere gestita secondo le seguenti dichiarazioni:

```
struct stackNode {
    char data;
    struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
```

Il programma deve essere composto dalla funzione main e da altre otto funzioni con i seguenti prototipi di funzioni.

Prototipo di funzione	Descrizione
<code>void convertToPostfix(char infix[], char postfix[]);</code>	Converte l'espressione infissa nella notazione postfissa.
<code>bool isOperator(char c);</code>	Restituisce true se c è un operatore; altrimenti, restituisce false. Ricordate che bool, true e false sono definiti nell'intestazione stdbool.h.
<code>int precedence(char operator1, char operator2);</code>	Restituisce -1, 0 o 1 per indicare se la precedenza di operator1 è, rispettivamente, minore, uguale o maggiore della precedenza di operator2.
<code>void push(StackNodePtr *topPtr, char value);</code>	Inserisce (push) un valore nella pila.
<code>char pop(StackNodePtr *topPtr);</code>	Estrae (pop) un valore dalla pila e restituisce quel valore.
<code>char stackTop(StackNodePtr topPtr);</code>	Restituisce il valore in cima alla pila senza estrarre dalla pila.
<code>bool isEmpty(StackNodePtr topPtr);</code>	Restituisce true se la pila è vuota (ovvero, topPtr è NULL); altrimenti, restituisce false.
<code>void printStack(StackNodePtr topPtr);</code>	Stampa la pila (questa funzione attraversa la lista collegata che implementa la pila, ma senza modificarla).

12.23 (Valutatore di espressioni postfisse) Scrivete un programma che valuti una corretta espressione postfissa come

6 2 + 5 * 8 4 / -

Il programma deve leggere un'espressione postfissa costituita da singole cifre e operatori memorizzata in un array di caratteri. Le espressioni postfisse non contengono parentesi; esse vengono eliminate durante la conversione da infisso a postfisso. Il programma deve analizzare l'espressione postfissa e valutarla usando le funzioni per le pile implementate precedentemente in questo capitolo.

- Aggiungete il carattere nullo ('\0') alla fine dell'espressione postfissa. Quando l'algoritmo incontra questo carattere nullo, la valutazione dell'espressione postfissa è completata.

2. Finché non si incontra il carattere nullo ('\0'), leggete l'espressione da sinistra a destra:

Se il carattere corrente è una cifra,

inserite (push) il suo valore intero nella pila. Il valore intero di un carattere cifra è il suo valore nell'insieme dei caratteri del computer meno il valore del carattere zero ('0') nello stesso insieme.

Altrimenti, se il carattere corrente è un operatore,

estraete (pop) i due elementi in cima alla pila e assegnateli alle variabili x e y;

calcolate y *operatore* x;

inserite il risultato del calcolo nella pila.

3. Quando nell'espressione si incontra il carattere nullo ('\0'), estraete il valore in cima alla pila. Questo è il risultato dell'espressione postfissa.

Questo algoritmo supporta solo operatori aritmetici binari. Pertanto nel *passo 2*, se l'operatore corrente è '/', la cima della pila è 2 e il successivo elemento nella pila è 8, allora si estra 2 e si assegna a x, si estra 8 e si assegna a y, si calcola 8 / 2 e si inserisce il risultato, 4, nella pila. Lo stesso vale per ogni operatore aritmetico binario.

In un'espressione sono permesse le seguenti operazioni aritmetiche:

- + addizione
- sottrazione
- * moltiplicazione
- / divisione

La pila deve essere gestita secondo le dichiarazioni seguenti:

```
struct stackNode {
    int data;
    struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
```

Il programma deve essere composto dalla funzione main e da altre sei funzioni con i seguenti prototipi di funzioni.

Prototipo di funzione	Descrizione
<code>int evaluatePostfixExpression(char *expr);</code>	Valuta l'espressione postfissa e ne restituisce il risultato.
<code>int calculate(int op1, int op2, char operator);</code>	Valuta l'espressione op1 operator op2 e ne restituisce il risultato.
<code>void push(StackNodePtr *topPtr, int value);</code>	Inserisce un valore nella pila.
<code>int pop(StackNodePtr *topPtr);</code>	Estrae un valore dalla pila e restituisce quel valore.
<code>bool isEmpty(StackNodePtr topPtr);</code>	Restituisce true se la pila è vuota (ovvero, topPtr è NULL); altrimenti, restituisce false.
<code>void printStack(StackNodePtr topPtr);</code>	Stampa la pila (questa funzione attraversa la lista collegata che implementa la pila, ma senza modificarla).

12.24 (Il linguaggio di programmazione Simple – Scrittura di programmi in Simple) Prima di costruire il compilatore, tratteremo un semplice ma potente linguaggio di alto livello, simile alle prime versioni del linguaggio di programmazione BASIC. Abbiamo chiamato questo linguaggio *Simple*. Ogni costrutto in Simple è composto da un numero di riga e da un’istruzione Simple. I numeri di riga devono essere in ordine crescente. Ogni istruzione inizia con uno dei seguenti comandi Simple: `rem`, `input`, `let`, `print`, `goto`, `if...goto` o `end`, descritti nella tabella sottostante. Simple valuta solo le espressioni intere, usando gli operatori `+`, `-`, `*` e `/`. Questi operatori hanno le stesse regole di precedenza del C. L’ordine di valutazione di un’espressione può essere cambiato usando le parentesi. L’Esercizio 12.27 suggerisce alcuni miglioramenti per il compilatore Simple. In molti casi, per effettuare questi miglioramenti, come l’aggiunta di funzionalità in virgola mobile, è necessario apportare modifiche anche alla macchina virtuale Simpletron.

Comando	Istruzione di esempio	Descrizione
<code>rem</code>	<code>50 rem this is a remark</code>	Il testo che segue il comando <code>rem</code> è solo a scopo di documentazione ed è ignorato: non viene generato codice SML.
<code>input</code>	<code>30 input x</code>	Visualizza un punto interrogativo per chiedere all’utente di inserire un singolo intero. Legge quell’intero dalla tastiera e lo memorizza in <code>x</code> .
<code>let</code>	<code>80 let u = 4 * (j - 7)</code>	Assegna a <code>u</code> il valore di <code>4 * (j - 7)</code> . Un’ espressione infissa arbitrariamente complessa può apparire a destra del segno di ugual.
<code>print</code>	<code>10 print w</code>	Visualizza il valore della singola variabile intera <code>w</code> .
<code>goto</code>	<code>70 goto 45</code>	Trasferisce il controllo del programma alla riga 45.
<code>if...goto</code>	<code>35 if i == z goto 80</code>	Confronta <code>i</code> e <code>z</code> per l’uguaglianza e trasferisce il controllo alla riga 80 se la condizione risulta vera; altrimenti, prosegue l’esecuzione con l’istruzione successiva.
<code>end</code>	<code>99 end</code>	Termina l’esecuzione del programma.

Altre regole del linguaggio Simple

Il linguaggio Simple segue anche le regole qui elencate.

- Il compilatore Simple **riconosce solo lettere minuscole**, quindi tutti i caratteri in un programma in Simple devono essere minuscoli.
- Il **nome delle variabili è composto da una singola lettera**. Non sono consentiti nomi di variabili con più caratteri, di conseguenza i programmi in Simple devono inserire la descrizione delle variabili nelle istruzioni `rem`.
- Simple usa **solo variabili intere**.
- Simple **non ha dichiarazioni di variabili**: è sufficiente la menzione di un nome di variabile in un programma perché avvenga la dichiarazione della variabile e la sua inizializzazione a zero.
- La sintassi di Simple **non consente la manipolazione di stringhe** (leggere o scrivere una stringa, confrontare stringhe, ecc.).
- Simple usa l’**istruzione di salto condizionato `if...goto`** e l’**istruzione di salto non condizionato `goto`** per modificare il flusso di controllo di un programma. Se la condizione nell’istruzione `if...goto` è vera, allora il controllo viene trasferito alla riga il cui numero è specificato. In un’istruzione `if...goto` sono considerati validi i seguenti operatori relazionali e di uguaglianza: `<`, `>`, `<=`, `>=`, `==` o `!=` (valgono le stesse regole di precedenza del C).

Il nostro compilatore assume che i programmi in Simple siano stati inseriti correttamente. L’Esercizio 12.27 vi chiede di modificare il compilatore affinché esegua il **controllo sugli errori di sintassi**.

Programmi in Simple di esempio

Prenderemo in considerazione diversi programmi in Simple che mostrano le caratteristiche di tale linguaggio. Il primo programma (Figura 12.5) legge due interi dalla tastiera, ne memorizza i valori nelle variabili *a* e *b*, poi calcola e stampa la loro somma (memorizzata nella variabile *c*).

```

10 rem    input two integers, then determine and print their sum
15 rem
20 rem    input the two integers
30 input a
40 input b
45 rem
50 rem    add integers and store result in c
60 let c = a + b
65 rem
70 rem    print the result
80 print c
90 rem    terminate program execution
99 end

```

Figura 12.5 Leggere due interi, poi calcolare e stampare la loro somma.

Il prossimo programma (Figura 12.6) determina e stampa il maggiore tra due interi. Gli interi vengono letti dalla tastiera e memorizzati nelle variabili *s* e *t*. L'istruzione *if...goto* verifica la condizione *s >= t*. Se la condizione è vera, allora il controllo viene trasferito alla riga 90, che stampa *s*. Altrimenti, il programma stampa *t*, poi trasferisce il controllo all'istruzione *end* nella riga 99, che termina il programma.

```

10 rem    input two integers, then determine and print the larger one
20 input s
30 input t
32 rem
35 rem    test if s is greater than or equal to t
40 if s >= t goto 90
45 rem
50 rem    t is greater than s, so print t
60 print t
70 goto 99
75 rem
80 rem    s is greater than or equal to t, so print s
90 print s
99 end

```

Figura 12.6 Leggere due interi, poi determinare e stampare il maggiore.

In Simple non esistono istruzioni di iterazione come *for*, *while* o *do...while* del C. Tuttavia, è possibile simularle usando le istruzioni *if...goto* e *goto*. Il programma della Figura 12.7 usa un ciclo controllato da sentinella per calcolare i quadrati di diversi interi. Ciascun intero è letto dalla tastiera e memorizzato nella variabile *j*. Se il valore inserito è il valore sentinella -9999, il controllo viene trasferito alla riga 99, che termina il programma. Altrimenti, a *k* viene assegnato il quadrato di *j*, *k* è stampato sullo schermo e il controllo passa alla riga 20, che legge l'intero successivo.

```

10 rem    calculate squares of integers until user enters -9999 sentinel to end
20 input j
23 rem
25 rem    test for sentinel value
30 if j == -9999 goto 99

```

```

33 rem
35 rem calculate square of j and assign result to k
40 let k = j * j
50 print k
53 rem
55 rem loop to get next j
60 goto 20
99 end

```

Figura 12.7 Calcolare quadrati di interi fino a quando l'utente non inserisce -9999 per terminare.

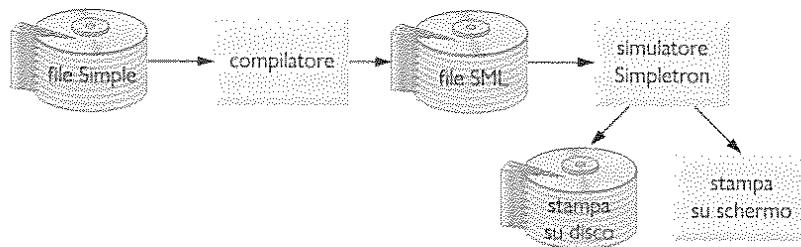
Scrittura di programmi in Simple

Utilizzando come guida i programmi di esempio delle Figure 12.5-12.7, scrivete programmi in Simple che eseguano le seguenti attività.

- Ricevere in input tre interi, determinare la loro media e stampare il risultato.
- Usare un ciclo controllato da sentinella per leggere 10 interi, poi calcolare e stampare la loro somma.
- Usare un ciclo controllato da contatore per leggere sette interi, alcuni positivi e altri negativi, poi calcolare e stampare la loro media.
- Ricevere in input una serie di interi, poi determinare e stampare il maggiore. Il primo input di interi indica quanti numeri devono essere elaborati.
- Ricevere in input 10 interi e stampare il minore.
- Calcolare e stampare la somma di tutti gli interi pari da 2 a 30.
- Calcolare e stampare il prodotto di tutti gli interi dispari da 1 a 9.

12.25 (Costruire un compilatore. Prerequisiti: aver completato gli Esercizi 7.28, 7.29, 12.22, 12.23 e 12.24)

Dopo aver presentato il linguaggio Simple (Esercizio 12.24), vediamo ora come costruire un compilatore Simple. La seguente figura riepiloga il processo per compilare un programma in Simple convertendolo in SML ed eseguirlo sul simulatore Simpletron:



Il compilatore legge un file che contiene un programma in Simple, lo **compila** in codice SML, che viene memorizzato in un file di testo, un'istruzione per riga. In seguito, il simulatore Simpletron **carica** il file SML nell'array della memoria di 100 elementi del Simpletron, esegue il programma e invia i risultati sia allo schermo che a un file. L'invio dell'output a un file serve a facilitare la stampa di una copia cartacea.

Il simulatore Simpletron che avete sviluppato nell'Esercizio 7.29 riceve l'input dalla tastiera, non da un file. Dovete pertanto modificare il Simpletron in modo che sia in grado di leggere da un file e quindi eseguire i programmi prodotti dal vostro compilatore Simple.

Il compilatore effettua i seguenti **due passaggi** su un programma in Simple per convertirlo in codice SML.

- Nel **primo passaggio** viene costruita una **tabella dei simboli** (discussa in dettaglio a breve). Il compilatore inserisce nella tabella dei simboli ogni **numero di riga**, **nome di variabile** e **costante** del programma in Simple. Ciascuno è memorizzato con il suo **tipo** e la sua **posizione** nel codice SML finale. Il **primo passaggio** produce inoltre le corrispondenti **istruzioni SML** per ogni istruzione Simple. Come vedrete, se il programma in Simple contiene istruzioni che trasferiscono il controllo a righe successive del programma, il **primo passaggio** produrrà un programma SML contenente alcune **istruzioni incomplete**.

- Il secondo passaggio individua e completa le istruzioni incomplete e memorizza il programma SML in un file. Il codice prodotto dal primo passaggio del compilatore è molto più esteso di quello risultante dal secondo passaggio.

Primo passaggio

Il compilatore inizia leggendo nella memoria la prima istruzione del programma in Simple. Il compilatore suddivide la riga in singoli **token** (ovvero “pezzi” di un’istruzione) per l’elaborazione e la compilazione. Per questa operazione si può usare la **funzione strtok**. Ricordate che ogni istruzione comincia con un **numero di riga** seguito da un **comando**. Durante la suddivisione in token del resto dell’istruzione, il compilatore pone nella **tabella dei simboli** ogni token identificato come un **numero di riga**, una **variabile** o una **costante**. Un **numero di riga** viene posto nella **tabella dei simboli** soltanto se è il **primo token** di un’istruzione (vedremo in seguito come si comporta il compilatore con i numeri di riga che sono i target di salti condizionati o non condizionati).

La tabella dei simboli **symbolTable** è un **array** di strutture **tableEntry** che rappresentano tutti i **simboli** del programma. Non c’è limite al numero di simboli che possono apparire nel programma, quindi **symbolTable** potrebbe essere di grandi dimensioni. Per il momento, create **symbolTable** come un array di 200 elementi. Potrete adattare le dimensioni dell’array quando il compilatore sarà funzionante.

La struttura **tableEntry** viene dichiarata come segue:

```
struct tableEntry {
    int symbol;
    char type; // 'C' (costante), 'L' (numero di riga), 'V' (variabile)
    int location; // da 00 a 99
};
```

Ogni struttura **tableEntry** contiene tre membri:

- **symbol** è un intero contenente la rappresentazione ASCII di una variabile (come già detto, i **nomi delle variabili** sono **caratteri singoli**), un **numero di riga** o una **costante** intera;
- **type** è un carattere che indica il tipo del simbolo: ‘C’ per una costante, ‘L’ per un numero di riga, o ‘V’ per una variabile;
- **location** contiene la **locazione nella memoria** del Simpletron (da 00 a 99) associata con il simbolo. La memoria del Simpletron è un array di 100 elementi interi in cui vengono memorizzati i **dati** e le **istruzioni SML**. Nel caso di un **numero di riga**, la **locazione** corrisponde all’elemento nell’**array della memoria** del Simpletron nel quale iniziano le istruzioni SML dell’istruzione in Simple. Nel caso di una **variabile** o una **costante**, la **locazione** corrisponde all’elemento nell’array della memoria del Simpletron nel quale è memorizzata la **variabile** o la **costante**. Le **variabili** e le **costanti** vengono allocate a partire dalla locazione 99 della memoria del Simpletron. La prima variabile o costante viene memorizzata nella locazione 99, la successiva nella locazione 98, e così via.

La **tabella dei simboli** svolge un ruolo fondamentale nella conversione dei programmi in Simple in linguaggio SML. Avete appreso nell’Esercizio 7.28 che un’**istruzione SML** è un **intero di quattro cifre** con segno composto da due parti: il **codice operativo** e l’**operando**. Il codice operativo è determinato dal comando Simple. Per esempio, il comando Simple **input** corrisponde al codice operativo **SML 10 (read)**, e il comando Simple **print** corrisponde al codice operativo **SML 11 (write)**. L’**operando** è una **locazione di memoria** che contiene i **dati** sui quali il **codice operativo** eseguirà il proprio compito. Per esempio, il codice operativo 10 legge un valore dalla tastiera e lo colloca nella **locazione di memoria** specificata dall’operando. Il compilatore effettua una ricerca nella **symbolTable** per determinare la **locazione nella memoria** del Simpletron di ciascun **simbolo** in modo che la **locazione** corrispondente possa essere usata per **completare le istruzioni SML**.

Il processo di compilazione di ciascuna istruzione Simple si basa su un particolare comando. Per esempio, dopo che il **numero di riga** di un’**istruzione rem** è stato inserito nella tabella dei simboli, la parte rimanente dell’istruzione viene ignorata dal compilatore (le istruzioni rem hanno soltanto scopo di documentazione) e non viene generato codice SML. Le istruzioni **input**, **print**, **goto** e **end** corrispondono alle **istruzioni SML read, write, branch** (a una **locazione** specificata) e **halt**. Il compilatore converte direttamente in SML le istruzioni che contengono questi comandi Simple. Un’**istruzione goto** potrebbe inizialmente contenere un

riferimento irrisolto se il numero di riga specificato facesse riferimento a un'istruzione successiva nel file del programma in Simple (in questo caso si parla di **riferimento in avanti**).

Quando un'istruzione **goto** viene compilata con un riferimento irrisolto, l'istruzione SML deve essere contrassegnata (*flagged*) per indicare che il secondo passaggio del compilatore deve completarla. I flag vengono memorizzati nell'array `int flags` di 100 elementi, ognuno dei quali è inizializzato a -1. Se la **locazione di memoria** alla quale un numero di riga fa riferimento non è ancora nota (ovvero, non è presente nella **tabella dei simboli**), il suo numero di riga viene memorizzato nell'array `flags` nell'elemento che ha lo stesso indice dell'istruzione incompleta. L'operando dell'istruzione incompleta è temporaneamente impostato a 00. Per esempio, un'istruzione di salto non condizionato (che faccia un riferimento in avanti) rimane impostata a +4000 fino al secondo passaggio del compilatore, che descriveremo a breve.

La compilazione di un'istruzione `if...goto` o `let` è più complicata rispetto alle altre, perché ciascuna di esse produce più di una istruzione SML. Per un'istruzione `if...goto`, il compilatore produce un codice che verifica la condizione ed eventualmente salta a un'altra riga. Il risultato del salto potrebbe essere un riferimento in avanti irrisolto. Ogni operatore relazionale e di uguaglianza del Simple può essere simulato usando istruzioni SML `branch zero` e `branch negative` (o anche una combinazione di entrambe).

Per un'istruzione `let`, il compilatore produce un codice che valuta un'espressione aritmetica infissa arbitrariamente complessa composta da operatori, nomi di variabili intere di una sola lettera, costanti intere ed eventualmente parentesi. Nell'espressione, ogni operando e operatore è separato da uno spazio. Gli Esercizi 12.22-12.23 hanno presentato l'algoritmo di conversione da infisso a postfisso e l'algoritmo per la valutazione dell'espressione postfissa usati dai compilatori per valutare le espressioni. Dovreste completare questi esercizi prima di iniziare la costruzione del vostro compilatore. Il compilatore converte ciascuna espressione da notazione infissa a notazione postfissa, quindi valuta l'espressione postfissa. Come vedrete, il compilatore genera in realtà istruzioni in linguaggio macchina durante il processo in cui valuta l'espressione postfissa.

In che modo il compilatore produce il linguaggio macchina per valutare un'espressione contenente variabili? L'algoritmo di valutazione dell'espressione postfissa contiene un "aggancio" (*hook*) che consente al compilatore di generare istruzioni SML invece di valutare effettivamente l'espressione. Per attivare questo "aggancio" nel compilatore, l'algoritmo per la valutazione dell'espressione postfissa deve essere modificato in modo che possa:

- ricercare nella tabella dei simboli ogni simbolo incontrato (ed eventualmente inserirlo);
- determinare la locazione di memoria corrispondente del simbolo;
- inserire in cima alla pila la locazione di memoria al posto del simbolo.

Nel momento in cui verrà incontrato un operatore nell'espressione postfissa, saranno estratte le due locazioni di memoria in cima alla pila e verrà prodotto il codice SML necessario per effettuare l'operazione usando le locazioni di memoria come operandi. Il risultato di ogni sottoespressione è memorizzato in una locazione di memoria temporanea e inserito nuovamente in cima alla pila, così che la valutazione dell'espressione postfissa possa continuare. Nel momento in cui la valutazione dell'espressione postfissa sarà completata, la locazione di memoria del risultato sarà l'unica locazione rimasta sulla pila. Essa verrà quindi estratta e saranno generate istruzioni SML per assegnare il risultato alla variabile a sinistra dell'istruzione `let`.

Secondo passaggio

Nel secondo passaggio il compilatore esegue due attività:

- risolvere ogni riferimento irrisolto;
- memorizzare il codice SML in un file.

La risoluzione di ciascun riferimento irrisolto avviene nel modo seguente.

1. Ricercate nell'array `flags` un riferimento irrisolto (ovvero, un elemento con un valore diverso da -1).
2. Individuate nell'array `symbolTable` la struttura che contiene il simbolo memorizzato nell'array `flags` (assicuratevi che il tipo del simbolo sia 'L' per i numeri di riga).
3. Inserite la locazione di memoria indicata nel membro della struttura `location` nell'istruzione con il riferimento irrisolto (ricordate che un'istruzione contenente un riferimento irrisolto ha come operando 00).
4. Ripetete i passi 1-3 finché non verrà raggiunta la fine dell'array `flags`.

Dopo il completamento del processo di risoluzione, il compilatore scriverà il codice SML generato su un file con un'istruzione SML per riga. Il Simpletron può leggere questo file ed eseguirne le istruzioni (ovviamente dopo che il simulatore sia stato modificato per leggere input da un file).

Un esempio completo

L'esempio seguente illustra la conversione completa in SML di un programma in Simple. Consideriamo un programma in Simple che riceva in input un intero, sommi i valori compresi tra 1 e quel determinato intero e stampi la somma risultante. Quindi, se l'utente inserisce in input 4, il programma eseguirà il calcolo $1 + 2 + 3 + 4$, il cui risultato è il valore 10, che verrà stampato. La Figura 12.8 mostra il programma e le istruzioni SML generate dal primo passaggio del compilatore. La Figura 12.9 mostra la tabella dei simboli costruita dal primo passaggio del compilatore. La Figura 12.10 mostra come il compilatore alloca la memoria del Simpletron verso il basso a partire dalla cella 99. Esamineremo a breve passo per passo la procedura con la quale il compilatore crea queste tabelle.

Programma in Simple	Locazione e istruzione SML	Descrizione
5 rem sum 1 to x	nessuna	rem ignorata
10 input x	00 +1099	legge x nella locazione 99
15 rem check y == x	nessuna	rem ignorata
20 if y == x goto 60	01 +2098	carica y (locazione 98) nell'accumulatore
	02 +3199	sottrae x (locazione 99) dall'accumulatore
	03 +4200	<i>branch zero</i> a una locazione irrisolta
25 rem increment y	nessuna	rem ignorata
30 let y = y + 1	04 +2098	carica y (locazione 98) nell'accumulatore
	05 +3097	aggiunge 1 (locazione 97) all'accumulatore
	06 +2196	memorizza nella locazione temporanea 96
	07 +2896	carica dalla locazione temporanea 96
	08 +2198	memorizza l'accumulatore in y (locazione 98)
35 rem add y to total t	nessuna	rem ignorata
40 let t = t + y	09 +2095	carica t (locazione 95) nell'accumulatore
	10 +3098	aggiunge y (locazione 98) all'accumulatore
	11 +2194	memorizza nella locazione temporanea 94
	12 +2094	carica dalla locazione temporanea 94
	13 +2195	memorizza l'accumulatore in t (locazione 95)
45 rem loop to y == x test	nessuna	rem ignorata
50 goto 20	14 +4001	salta alla locazione 01
55 rem output result	nessuna	rem ignorata
60 print t	15 +1195	stampa t (locazione 95) sullo schermo
99 end	16 +4300	termina l'esecuzione

Figura 12.8 Istruzioni SML generate dopo il primo passaggio del compilatore.

Simbolo	Tipo	Locazione
5	L	00
10	L	00
'x'	V	99
15	L	01
20	L	01
'y'	V	98
25	L	04
30	L	04
1	C	97
<i>Allocato temporaneamente a 96</i>		
35	L	09
40	L	09
't'	V	95
<i>Allocato temporaneamente a 94</i>		
45	L	14
50	L	14
55	L	15
60	L	15
99	L	16

Figura 12.9 Tabella dei simboli per il programma della Figura 12.8.

Contatore di dati	Valore	Tipo
...		
93		Successiva cella di memoria del Simpletron da allocare
94	nessuno	<i>Variabile temporanea</i>
95	't'	Variabile
96	nessuno	<i>Variabile temporanea</i>
97	1	Costante
98	'y'	Variabile
99	'x'	Variabile

Figura 12.10 Il compilatore esegue l'allocazione nella memoria del Simpletron verso il basso a partire dall'ultima cella della memoria (99).

La maggior parte delle istruzioni Simple vengono convertite direttamente in una singola istruzione SML. In questo programma, le eccezioni riguardano le istruzioni `rem`, l'istruzione `if...goto` della riga 20 e le istruzioni `let` delle righe 30 e 40. I commenti non vengono tradotti in linguaggio macchina. Tuttavia, i numeri di riga relativi ai commenti sono comunque inseriti nella tabella dei simboli, nel caso che siano stati puntati da un'istruzione `goto` o `if...goto`.

La riga 20 del programma specifica che se la condizione `y == x` è vera, il controllo del programma viene trasferito alla riga 60. Dato che la riga 60 apparirà *più avanti* nel programma, il **primo passaggio** del compilatore non ha ancora inserito 60 nella tabella dei simboli (i numeri di riga vengono collocati nella tabella dei simboli solo quando compaiono come primo token in un'istruzione che è stata elaborata dal compilatore). Di conseguenza, a questo punto non è possibile determinare l'operando dell'istruzione SML `branch zero` alla locazione 03 nell'array delle istruzioni SML. Il compilatore colloca il 60 nella locazione 03 dell'array `flags` per indicare che il secondo passaggio dovrà completare questa istruzione.

È necessario tenere traccia della locazione dell'istruzione successiva nell'array SML poiché **non esiste una corrispondenza uno a uno tra le istruzioni Simple e le istruzioni SML**. Per esempio, il compilatore converte l'istruzione `if...goto` della riga 20 in *tre* istruzioni SML. Ogni volta che viene generata un'istruzione, dobbiamo incrementare il contatore di istruzioni alla locazione successiva dell'array SML. Le dimensioni limitate della memoria del Simpletron possono costituire un problema per i programmi in Simple con un numero elevato di istruzioni, variabili e costanti. Potrebbe accadere che il compilatore esaurisca la memoria del Simpletron. Per tenere sotto controllo questa possibilità, il vostro programma deve contenere un **contatore di dati** che tenga traccia della locazione nella quale la variabile o la costante successiva verrà memorizzata nell'array SML. Se il valore del contatore di istruzioni è maggiore del valore del contatore dei dati, l'array SML è pieno. In questo caso, il processo di compilazione deve terminare e il compilatore visualizzerà il **messaggio di errore** "memoria esaurita".

Analisi dettagliata del primo passaggio del processo di compilazione

Analizziamo passo per passo il processo di compilazione per il programma in Simple nella Figura 12.8. Il compilatore legge la prima riga del programma:

5 rem sum 1 to x

Il primo token nell'istruzione (il numero di riga) viene determinato usando la funzione `strtok` (nel Capitolo 8 sono state trattate le funzioni per la manipolazione di stringhe del C). Il token restituito da `strtok` viene convertito in un intero usando la funzione `atoi`, in modo che il simbolo 5 possa essere collocato nella tabella dei simboli. Se il simbolo non viene trovato, viene inserito nella tabella. Dato che il programma è all'inizio e questa è la sua prima riga, la tabella non contiene ancora alcun simbolo. Quindi 5 viene inserito nella tabella dei simboli come tipo L (numero di riga) e gli viene assegnata la prima locazione nell'array della memoria SML (00). Sebbene questa riga rappresenti un commento, viene comunque allocato uno spazio nella tabella dei simboli per il numero di riga (nel caso in cui sia puntato da un'istruzione `goto` o `if...goto`). Se un programma salta al numero di riga di un'istruzione `rem`, il controllo riprende dalla prima istruzione eseguibile dopo l'istruzione `rem`. Per un'istruzione `rem` non viene generata alcuna istruzione SML, pertanto il contatore di istruzioni non viene incrementato.

In seguito, il compilatore suddivide in token l'istruzione

10 input x

Il numero di riga 10 è collocato nella tabella dei simboli come tipo L e gli viene assegnata la prima locazione nell'array SML (00): il contatore di istruzioni è ancora 00 perché la prima istruzione del programma era un commento. Il comando `input` indica che il token successivo è una variabile (soltanto una variabile può comparire come argomento in un'istruzione `input`). Dato che `input` corrisponde direttamente a un codice operativo SML, il compilatore deve semplicemente determinare la locazione di `x` nell'array SML. Il simbolo `x` non viene trovato nella tabella e, pertanto, viene inserito nella tabella dei simboli come la **rappresentazione ASCII** di `x`, gli viene dato il tipo V (variabile) e assegnata la locazione 99 nell'array SML. La memorizzazione dei dati parte da 99 e viene allocata verso il basso: 98, 97 e così via. Ora può essere generato il codice SML per questa istruzione. Il codice operativo 10 (il codice operativo SML `read`) è moltiplicato per 100, e al risultato viene sommata la locazione di `x` (come determinata nella tabella dei simboli), completando così l'istruzione +1099,

che viene quindi memorizzata nell'array SML alla locazione 00. Il contatore di istruzioni viene incrementato di 1 dato che è stata generata una singola istruzione SML.

In seguito, il compilatore suddivide in token l'istruzione

```
15 rem check y == x
```

Il numero di riga 15 viene ricercato nella tabella dei simboli ma non viene trovato. Il numero di riga è inserito come tipo L e gli viene assegnata la locazione successiva nell'array SML (01). Come già detto, le istruzioni rem non generano codice, quindi il contatore di istruzioni non viene incrementato.

In seguito, il compilatore suddivide in token l'istruzione

```
20 if y == x goto 60
```

Il numero di riga 20 viene inserito nella tabella dei simboli come tipo L e gli viene assegnata la locazione successiva nell'array SML, 01. Il comando if indica che deve essere valutata una condizione. La variabile y non viene trovata nella tabella dei simboli, quindi viene inserita e le vengono assegnati il tipo V e la locazione SML 98. A seguire, vengono generate le istruzioni SML per valutare la condizione. Dato che non esiste alcuna diretta corrispondenza in SML per l'istruzione if...goto, quest'ultima dovrà essere simulata eseguendo un calcolo con x e y e trasferendo il controllo in base al risultato. Se y è uguale a x, il risultato della sottrazione di x da y sarà zero. Pertanto, l'istruzione **branch-zero** SML può essere usata con il risultato del calcolo per simulare l'istruzione if...goto.

Il primo passo richiede che y venga caricata (dalla locazione SML 98) nell'accumulatore, generando così l'istruzione 01 +2098. Poi viene sottratta x dall'accumulatore, il che genera l'istruzione 02 +3199. Il valore nell'accumulatore potrebbe essere zero, positivo o negativo. Poiché l'operatore è ==, useremo l'istruzione **branch zero**. In primo luogo, viene ricercata nella tabella dei simboli la locazione di destinazione per il salto (60), che però non viene trovata. Allora 60 viene inserito nell'array flags alla locazione 03, e viene generata l'istruzione 03 +4200. Non possiamo aggiungere la locazione del salto perché non abbiamo ancora assegnato una locazione alla riga 60 nell'array SML (questa locazione verrà risolta più avanti). Il contatore di istruzioni viene incrementato a 04.

Il compilatore procede con l'istruzione

```
25 rem increment y
```

Il numero di riga 25 viene inserito nella tabella dei simboli come tipo L e gli viene assegnata la locazione SML 04. Il contatore di istruzioni non viene incrementato.

Quando viene suddivisa in token l'istruzione

```
30 let y = y + 1
```

il numero di riga 30 viene inserito nella tabella dei simboli come tipo L e gli viene assegnata la locazione SML 04. Il comando let indica che la riga è un'istruzione di assegnazione. In primo luogo, tutti i simboli della riga vengono inseriti nella tabella dei simboli (se non sono già presenti). La costante intera 1 viene inserita come tipo C e le viene assegnata la locazione SML 97. Successivamente, la parte destra dell'istruzione di assegnazione viene convertita da notazione infissa a postfissa, e viene poi valutata l'espressione postfissa (y 1 +). Il simbolo y viene individuato nella tabella dei simboli, e la sua corrispondente locazione di memoria, 98, viene inserita in cima alla pila. Anche il simbolo 1 viene individuato nella tabella dei simboli, e la sua corrispondente locazione di memoria, 97, viene inserita in cima alla pila. Quando incontra l'operatore +, la funzione di valutazione delle espressioni postfisse estrae un valore dalla pila e lo assegna all'operando di destra dell'operatore, poi nuovamente estrae un valore dalla pila e lo assegna all'operando di sinistra dell'operatore, infine genera le istruzioni SML

```
04 +2098 (load y)
05 +3097 (add 1)
```

Il risultato dell'espressione viene memorizzato in una locazione di memoria temporanea (96) con l'istruzione

```
06 +2196 (store temporary)
```

e la locazione temporanea è *inserita in cima alla pila*. Ora l'espressione è stata valutata e il risultato deve essere memorizzato nella variabile y dell'istruzione let. Pertanto la locazione temporanea viene caricata nell'accumulatore, e questo viene memorizzato in y con le istruzioni

```
07 +2096 (load temporary)
08 +2198 (store y)
```

Osservate che alcune di queste istruzioni SML (memorizzare l'accumulatore nella locazione temporanea 96 e subito dopo caricare nuovamente l'accumulatore dalla locazione 96) sembrano essere **ridondanti**. L'eliminazione della ridondanza è un esempio dell'**ottimizzazione del compilatore**, di cui discuteremo a breve.

Quando il compilatore suddivide in token l'istruzione

```
35 rem add y to total
```

inserisce il numero di riga 35 nella tabella dei simboli come tipo L e gli assegna la locazione 09.

L'istruzione seguente è simile alla riga 30:

```
40 let t = t + y
```

La variabile t è inserita nella tabella dei simboli come tipo V e le viene assegnata la locazione SML 95. Le istruzioni seguono la stessa logica e lo stesso formato della riga 30, e vengono generate le istruzioni 09 +2095, 10 +3098, 11 +2194, 12 +2094 e 13 +2195. Il risultato di t + y viene assegnato alla locazione temporanea 94 prima di essere assegnato a t (95). Anche le istruzioni 11 e 12 sembrano essere **ridondanti**. Come anticipato, discuteremo a breve dell'ottimizzazione.

L'istruzione

```
45 rem loop to y == x test
```

è un commento, quindi la riga 45 è inserita nella tabella dei simboli come tipo L e le viene assegnata la locazione SML 14.

L'istruzione

```
50 goto 20
```

trasferisce il controllo alla riga 20. Il numero di riga 50 è inserito nella tabella dei simboli come tipo L e gli viene assegnata la locazione SML 14. L'**equivalente di goto in SML** è l'**istruzione di salto non condizionato** (40) che trasferisce il controllo a una specifica locazione SML. Il compilatore ricerca la riga 20 nella tabella dei simboli e trova la sua corrispondenza con la locazione SML 01. Il codice operativo (40) è moltiplicato per 100, e al risultato viene sommata la locazione 01, generando così l'istruzione +4001 alla locazione 14.

L'istruzione

```
55 rem output result
```

è un commento, quindi la riga 55 viene inserita nella tabella dei simboli come tipo L e le viene assegnata la locazione SML 15.

L'istruzione

```
60 print t
```

è un'istruzione di output. Il numero di riga 60 è inserito nella tabella dei simboli come tipo L e gli viene assegnata la locazione SML 15. L'**equivalente di print in SML** è il **codice operativo 11 (write)**. La locazione della variabile t viene determinata dalla tabella dei simboli, quindi sommata al codice operativo moltiplicato per 100. Viene così generata l'istruzione +1195 alla locazione 15.

L'istruzione

```
99 end
```

è la riga finale del programma. Il numero di riga 99 è inserito nella tabella dei simboli come tipo L e gli viene assegnata la locazione SML 16. Il comando end genera l'istruzione SML +4300 (43 corrisponde a halt in SML), che viene scritta come istruzione finale nell'array della memoria SML. L'istruzione halt non ha operandi. Riuscite a pensare a un motivo valido per consentire la presenza di un operando nell'istruzione halt?

Secondo passaggio del processo di compilazione

Iniziamo il *secondo passaggio* del compilatore ricercando nell'**array flags** valori diversi da -1. La locazione 03 contiene 60, pertanto il compilatore capisce che l'istruzione 03 è incompleta. Il compilatore procede a completarla cercando 60 nella tabella dei simboli, determinandone la locazione e aggiungendola all'istruzione incompleta. In questo caso, la ricerca determina che la riga 60 corrisponde alla locazione SML 15, quindi viene generata alla locazione 03 l'istruzione completa +4215, che sostituisce +4200. Il programma in Simple è stato ora compilato con successo.

Costruire il proprio compilatore

Per costruire il compilatore, dovete eseguire le seguenti attività:

- Modificate il programma del simulatore Simpletron che avete scritto nell'Esercizio 7.29 in modo che riceva il suo input da un file specificato dall'utente (vedi Capitolo 11). Il simulatore deve anche inviare i risultati a un file, nello stesso formato dell'output sullo schermo.
- Modificate l'algoritmo di conversione da infisso a postfisso dell'Esercizio 12.22 in modo che sia in grado di elaborare **operandi interi a più cifre** e **operandi corrispondenti a nomi di variabili composti da una singola lettera**. È possibile usare le funzioni della Libreria Standard `strtok` per individuare tutte le costanti e le variabili in un'espressione e `atoi` per convertire le costanti da stringhe a interi. La rappresentazione dei dati dell'espressione postfissa deve essere modificata per supportare i nomi delle variabili e le costanti intere.
- Modificate l'algoritmo per la valutazione dell'espressione postfissa in modo che sia in grado di elaborare **operandi interi a più cifre** e **operandi corrispondenti a nomi di variabili composti da una singola lettera**. L'algoritmo ora deve anche implementare l'"aggancio" (*hook*) discusso in precedenza in modo che generi istruzioni SML invece di valutare direttamente l'espressione. È possibile usare le funzioni della Libreria Standard `strtok` per individuare tutte le costanti e le variabili in un'espressione e `atoi` per convertire le costanti da stringhe a interi. La rappresentazione dei dati dell'espressione postfissa deve essere modificata per supportare i nomi delle variabili e le costanti intere.
- Costruite il compilatore. Incorporate la *parte b* e la *parte c* per valutare le espressioni nelle istruzioni `let`. Il vostro programma deve contenere una funzione che esegue il *primo passaggio* del compilatore e una funzione che esegue il *secondo passaggio*.

12.26 (Ottimizzazione del compilatore Simple) Quando un programma è compilato e convertito in SML, viene generato un insieme di istruzioni. Alcune combinazioni di istruzioni spesso si ripetono, spesso in triplette chiamate **produzioni**. Una produzione di norma è composta da tre istruzioni, quali **load**, **add** e **store**. Per esempio, la Figura 12.11 mostra cinque delle istruzioni SML generate durante la compilazione del programma della Figura 12.8. Le prime tre istruzioni compongono la produzione che aggiunge 1 a y. Le istruzioni 06 e 07 memorizzano il valore dell'accumulatore nella locazione temporanea 96, poi caricano nuovamente il valore da quella posizione nell'accumulatore così che l'istruzione 08 possa memorizzare tale valore nella locazione 98. Spesso una produzione è seguita da un'istruzione **load** per la stessa locazione che è stata appena usata per la memorizzazione. Questo codice può essere *ottimizzato* eliminando l'istruzione **store** e la conseguente istruzione **load** che operano sulla stessa locazione di memoria. Con questa ottimizzazione si ottiene un risparmio di memoria del 25% per il programma SML e un aumento della sua velocità di esecuzione. La Figura 12.12 mostra il **codice SML ottimizzato** per il programma della Figura 12.8. Ci sono **quattro istruzioni in meno nel codice ottimizzato**. Modificate il compilatore in modo che esegua l'ottimizzazione vista in questo esercizio.

04 +2098	(load)
05 +3097	(add)
06 +2196	(store)
07 +2096	(load)
08 +2198	(store)

Figura 12.11 Codice non ottimizzato per il programma della Figura 12.8.

Programma in Simple	Locazione e Istruzione SML	Descrizione
5 rem sum 1 to x	nessuna	rem ignorata
10 input x	00 +1099	legge x nella locazione 99
15 rem check y == x	nessuna	rem ignorata
20 if y == x goto 60	01 +2098	carica y (98) nell'accumulatore
	02 +3199	sottrae x (99) dall'accumulatore
	03 +4211	se zero, salta alla locazione 11
25 rem increment y	nessuna	rem ignorata
30 let y = y + 1	04 +2098	carica y nell'accumulatore
	05 +3097	aggiunge 1 (97) all'accumulatore
	06 +2198	memorizza l'accumulatore in y (98)
35 rem add y to total	nessuna	rem ignorata
40 let t = t + y	07 +2096	carica t dalla locazione (96)
	08 +3098	aggiunge y (98) all'accumulatore
	09 +2196	memorizza l'accumulatore in t (96)
45 rem loop to y == x test	nessuna	rem ignorata
50 goto 20	10 +4001	salta alla locazione 01
55 rem output result	nessuna	rem ignorata
60 print t	11 +1196	stampa t (96) sullo schermo
99 end	12 +4300	termina l'esecuzione

Figura 12.12 Codice ottimizzato per il programma della Figura 12.8.

12.27 (Miglioramenti al compilatore Simple) Apportate le modifiche che seguono al compilatore Simple. In alcuni casi saranno necessarie anche variazioni al programma del simulatore Simpletron che avete scritto nell'Esercizio 7.29. Molte di queste modifiche sono particolarmente impegnative e potrebbero richiedere uno sforzo notevole.

- Modificate la memoria del Simpletron in modo che abbia **1.000 celle (000-999)**. Modificate il compilatore per generare linguaggio macchina appropriato per la memoria di 1.000 elementi del Simpletron.
- Consentite al compilatore di elaborare valori in virgola mobile** oltre agli interi. Anche il simulatore Simpletron deve essere modificato per elaborare valori in virgola mobile.
- Aggiungete il supporto per l'operatore meno unario** per specificare valori interi negativi.
- Consentite l'uso dell'**operatore modulo (%)** nelle istruzioni let. Modificate il linguaggio macchina del Simpletron per includere l'istruzione modulo.
- Consentite l'**elevamento a potenza** nelle istruzioni let usando il simbolo **^ come operatore di elevamento a potenza**. Modificate il linguaggio macchina del Simpletron per includere l'istruzione di elevamento a potenza.
- Consentite al compilatore di riconoscere lettere minuscole e maiuscole** nelle istruzioni Simple. In questo modo, x e X saranno gestite come variabili differenti. Non sono necessarie modifiche al simulatore Simpletron.

- g) Consentite alle istruzioni **input** di leggere valori per variabili multiple, come **input x, y**. Non sono necessarie modifiche al simulatore Simpletron.
- h) Consentite al compilatore di stampare valori multipli con una singola istruzione **print**, come **print a, b, c**. Questa istruzione stamperà i valori delle variabili, separati l'uno dall'altro con uno spazio. Non sono necessarie modifiche al simulatore Simpletron.
- i) Consentite che l'operando dell'istruzione **print** sia un'espressione infissa.
- j) Aggiungete al compilatore le funzionalità di **controllo della sintassi**, in modo che vengano visualizzati messaggi di errore qualora fossero riscontrati errori di sintassi in un programma in Simple. Non sono necessarie modifiche al simulatore Simpletron.
- k) Consentite l'uso di array di interi. Non sono necessarie modifiche al simulatore Simpletron.
- l) Consentite l'uso di subroutine specificate dai comandi **Simple gosub e return**. Il comando **gosub** passa il controllo del programma a una subroutine, e il comando **return** restituisce il controllo all'istruzione successiva dopo il **gosub**. Si tratta di un meccanismo simile alla chiamata di funzione del C. La stessa subroutine può essere chiamata da molti **gosub** distribuiti lungo tutto il programma. Non sono necessarie modifiche al simulatore Simpletron.
- m) Consentite l'uso di strutture di iterazione della forma

```
for x = 2 to 10
    rem Simple statements
next
```

Questa istruzione **for** itera da 2 a 10 con un incremento predefinito di 1. Non sono necessarie modifiche al simulatore Simpletron.

- n) Consentite l'uso di strutture di iterazione della forma

```
for x = 2 to 10 step 2
    rem Simple statements
next
```

Questa istruzione **for** itera da 2 a 10 con un incremento di 2. La riga **next** segna la fine del corpo dell'istruzione **for**. Non sono necessarie modifiche al simulatore Simpletron.

CAPITOLO

13

Sommario del capitolo

- 13.1 Introduzione
- 13.2 Efficienza degli algoritmi:
 \mathcal{O} grande
- 13.3 Ordinamento per selezione
- 13.4 Ordinamento per inserzione
- 13.5 Caso pratico: visualizzazione
dell'algoritmo di ordinamento
per fusione ad alte
prestazioni
- 13.6 Riepilogo

Pensare come un informatico: algoritmi di ordinamento e \mathcal{O} grande

Obiettivi

- Ordinare un array usando l'algoritmo di ordinamento per selezione.
- Ordinare un array usando l'algoritmo di ordinamento per inserzione.
- Ordinare un array usando l'algoritmo ricorsivo di ordinamento per fusione.
- Comprendere l'efficienza degli algoritmi di ordinamento ed esprimere nella notazione \mathcal{O} grande.
- Esplorare (negli esercizi) altri tipi di ordinamento ricorsivo, tra cui il quicksort e un ordinamento ricorsivo per selezione.
- Esplorare (negli esercizi) l'algoritmo di ordinamento ad alte prestazioni bucket sort.

13.1 Introduzione

Nel Capitolo 6 avete appreso che l'ordinamento mette i dati in ordine crescente o decrescente sulla base di una o più chiavi di ordinamento. Introduciamo in questo capitolo gli algoritmi di ordinamento per selezione, per inserzione e per fusione (quest'ultimo più efficiente ma più complesso). Introduciamo la **notazione “ \mathcal{O} grande”**, usata per stimare il tempo di esecuzione per un algoritmo nel caso peggiore (una misura di quanto sforzo può essere richiesto a un algoritmo per risolvere un problema).

Riguardo all'ordinamento di array, è importante capire che il risultato finale sarà lo stesso a prescindere dall'algoritmo di ordinamento utilizzato. La scelta di un algoritmo influenza solo sul tempo di esecuzione e sull'uso della memoria nel programma. Gli algoritmi di ordinamento per selezione e per inserzione che qui studiamo sono facili da programmare ma inefficienti. Il terzo algoritmo – l'ordinamento ricorsivo per fusione – è più efficiente ma più difficile da programmare.

Gli esercizi presentano altri due ordinamenti ricorsivi: il quicksort e una versione ricorsiva dell'ordinamento per selezione. Un altro esercizio presenta il bucket sort, il quale raggiunge alte prestazioni attraverso un uso intelligente di parecchia più memoria rispetto agli altri ordinamenti che esaminiamo.

✓ Autovalutazione

1. *(Completare)* L'ordinamento mette i dati in ordine crescente o decrescente in base a una o più _____ di ordinamento.
- Risposta:** chiavi.
2. *(Scelta multipla)* Quale tra le seguenti affermazioni è *falsa*?
 - a) La notazione O grande stima il tempo di esecuzione per un algoritmo nel caso migliore (una misura di quanto sforzo può essere richiesto a un algoritmo per risolvere un problema).
 - b) Nell'ordinamento, l'array ordinato sarà lo stesso a prescindere dall'algoritmo di ordinamento utilizzato.
 - c) Il tipo di algoritmo di ordinamento scelto influenza solo sul tempo di esecuzione e sull'uso della memoria nel programma.
 - d) Gli algoritmi di ordinamento per selezione e per inserzione sono facili da programmare ma inefficienti.
L'ordinamento ricorsivo per fusione è più efficiente, ma più difficile da programmare.

Risposta: a) è *falsa*. In realtà, la notazione O grande stima il tempo di esecuzione per un algoritmo nel *caso peggiore*.

❖ 13.2 Efficienza degli algoritmi: O grande

La notazione O grande è una maniera per descrivere lo sforzo richiesto a un algoritmo e indica quanta mole di lavoro esso deve svolgere per risolvere un problema. Per gli algoritmi di ricerca e ordinamento, tutto ciò dipende principalmente dal numero di elementi che sono presenti nei dati. In questo capitolo usiamo la notazione O grande per descrivere i tempi di esecuzione nel caso peggiore dei vari algoritmi di ordinamento.

13.2.1 Algoritmi O(1)

Supponiamo di avere un algoritmo che controlla se il primo elemento di un array è uguale al secondo. Se l'array ha 10 elementi, questo algoritmo richiede un confronto. Se l'array ha 1.000 elementi, l'algoritmo richiede ancora un confronto. L'algoritmo è completamente indipendente dal numero di elementi presenti nell'array. Diciamo che questo algoritmo ha **tempo di esecuzione costante**, che in notazione O grande si indica con **O(1)**. O(1) si legge anche “ordine uno”. Un algoritmo O(1) non richiede necessariamente un solo confronto. O(1) significa che il numero di confronti è *costante*, cioè che non cresce al crescere della dimensione dell'array. Anche un algoritmo che controlla se il primo elemento di un array è uguale ai tre successivi è O(1), pur richiedendo tre confronti.

13.2.2 Algoritmi O(n)

Un algoritmo che controlla se il primo elemento di un array è uguale a un *qualsiasi* altro elemento richiede al più $n - 1$ confronti, dove n è il numero di elementi dell'array. Se l'array ha 10 elementi, questo algoritmo richiederà al massimo nove confronti. Se l'array ha 1.000 elementi, questo algoritmo richiederà fino a 999 confronti.

Al crescere di n , nell'espressione $n - 1$ la parte “dominante” sarà n e sottrarre 1 diventerà irrilevante. La notazione O grande serve a sottolineare questi termini dominanti e a ignorare i termini che diventano insignificanti al crescere di n . Quindi, un algoritmo che richiede $n - 1$ confronti in totale viene detto **O(n)**. Si dice che un algoritmo O(n) ha **tempo di esecuzione lineare**. O(n) si legge spesso come “nell'ordine di n ” o più semplicemente “ordine n ”.

13.2.3 Algoritmi O(n^2)

Supponiamo di avere un algoritmo che controlla se *tutti* gli elementi di un array sono duplicati all'interno dell'array. L'algoritmo confronta il primo elemento con ogni altro elemento nell'array. L'algoritmo poi confronta il secondo elemento con tutti gli elementi dell'array eccetto il primo (sono già stati confrontati). Poi, l'algoritmo confronta il terzo elemento con tutti gli elementi dell'array eccetto il primo e il secondo. Alla fine, l'algoritmo farà $(n - 1) + (n - 2) + \dots + 2 + 1$ ovvero $n^2/2 - n/2$ confronti. Al crescere di n , il termine con n^2 domina l'espressione, mentre quello con n diventa irrilevante. Di nuovo, la notazione O grande mette in evidenza

il termine con n^2 , ignorando $n^2/2$. Ma come vedremo presto, i fattori costanti vengono omessi nella notazione O grande.

La notazione O grande riguarda la relazione che intercorre tra il numero degli elementi da elaborare e il tempo di esecuzione dell'algoritmo. Supponiamo che un algoritmo richieda n^2 confronti. Su quattro elementi, l'algoritmo richiederà 16 confronti; su otto elementi, 64 confronti. Con questo algoritmo, se *raddoppio* i dati in ingresso, *quadruplico* il numero di confronti. Consideriamo un algoritmo simile, che richieda $n^2/2$ confronti. Su quattro elementi, l'algoritmo richiederà 8 confronti; su otto elementi, 32 confronti. Di nuovo, *raddoppiare* i dati *quadruplica* i confronti. Entrambi gli algoritmi crescono come il quadrato di n . La notazione O grande ignora la parte costante ed entrambi gli algoritmi vengono considerati $O(n^2)$. Si dice che il **tempo di esecuzione è quadratico**. $O(n^2)$ viene letto “nell'ordine di n quadro” o più semplicemente “ordine n quadro”.

Quando n è piccolo, l'influsso degli algoritmi $O(n^2)$ (in esecuzione sui personal computer moderni con un miliardo di operazioni al secondo) sulle prestazioni non è evidente, ma al crescere di n , il calo delle prestazioni comincia a notarsi. Un algoritmo $O(n^2)$ che elabora un array di un milione di elementi richiederà mille miliardi di “operazioni” (ognuna delle quali potrebbe richiedere diverse istruzioni in linguaggio macchina) e potrebbe essere necessaria qualche ora per l'esecuzione. Un array con un miliardo di elementi richiederebbe un miliardo di miliardi di operazioni, un numero così grande che l'algoritmo potrebbe impiegare decenni! Come vedremo in questo capitolo, gli algoritmi $O(n^2)$ sono facili da scrivere. Vedremo anche algoritmi con misure O grandi più favorevoli. Per creare tali algoritmi più efficienti, spesso sono richiesti un po' di furbizia e di lavoro in più, ma vale la pena impegnarsi per ottenere queste migliori prestazioni, specialmente nelle situazioni in cui n diventa grande o quando l'algoritmo viene integrato in programmi più complessi.

✓ Autovalutazione

1. (*Vero/Falso*) Gli algoritmi $O(n^2)$, in esecuzione sui personal computer moderni con un miliardo di operazioni al secondo non influiscono in modo evidente sulle prestazioni.

Risposta: *Falso*. In realtà, quando n è *piccolo*, l'influsso degli algoritmi $O(n^2)$ sulle prestazioni non è evidente. Ma al crescere di n , il calo delle prestazioni comincia a notarsi, persino sui potenti sistemi attuali.

2. (*Completare*) La notazione O grande riguarda la relazione che intercorre tra _____ e il tempo di esecuzione dell'algoritmo.

Risposta: il numero degli elementi da elaborare.

3. (*Vero/Falso*) Un algoritmo $O(1)$ richiede un solo confronto.

Risposta: *Falso*. $O(1)$ significa che il numero di confronti è *costante*. L'algoritmo potrebbe richiedere confronti multipli, ma il numero dei confronti non cresce al crescere della dimensione dell'array.

4. (*Completare*) Si dice che un algoritmo $O(n)$ ha tempo di esecuzione _____.

Risposta: lineare.

5. (*Completare*) Si dice che un algoritmo $O(n^2)$ ha tempo di esecuzione _____.

Risposta: quadratico.

13.3 Ordinamento per selezione

L'ordinamento per selezione (*selection sort*) è un algoritmo semplice, ma inefficiente.

- La prima iterazione dell'algoritmo seleziona l'elemento più piccolo nell'array e lo scambia con il primo elemento.
- La seconda iterazione seleziona il secondo elemento più piccolo (che è il più piccolo di quelli restanti) e lo scambia con il secondo elemento.
- L'algoritmo continua fino a che l'ultima iterazione seleziona il secondo elemento più grande e lo scambia con il penultimo. L'elemento più grande rimane così l'ultimo.

Dopo l' i -esima iterazione, gli i elementi più piccoli saranno sistemati in ordine crescente nelle prime i posizioni dell'array.

Come esempio, consideriamo l'array:

34 56 4 10 77 51 93 30 5 52

L'ordinamento per selezione per prima cosa determina l'elemento più piccolo (4) in questo array, poi lo scambia con il valore nell'elemento 0 (34), producendo:

4 56 34 10 77 51 93 30 5 52

L'ordinamento per selezione poi determina il valore rimanente più piccolo partendo dall'elemento 1, che è il valore 5 nell'elemento 8. L'algoritmo scambia 5 con il valore 56 nell'elemento 1, producendo:

4 5 34 10 77 51 93 30 56 52

Alla terza iterazione, l'ordinamento per selezione determina il successivo valore più piccolo (10 nell'elemento 3) e lo scambia con 34 nell'elemento 2, producendo:

4 5 10 34 77 51 93 30 56 52

Il processo continua finché dopo nove iterazioni l'array non è completamente ordinato, come in:

4 5 10 30 34 51 52 56 77 93

Dopo la prima iterazione, l'elemento più piccolo si trova nell'elemento 0. Dopo la seconda iterazione, i due elementi più piccoli sono ordinati negli elementi 0 e 1. Dopo la terza iterazione, i tre elementi più piccoli sono ordinati negli elementi 0-2, e così via.

13.3.1 Implementazione dell'ordinamento per selezione

Il programma della Figura 13.1 implementa l'algoritmo di ordinamento per selezione. Le righe 18-20 riempiono `array` con 10 valori `int` casuali. La funzione `main` stampa l'array non ordinato, passa `array` alla funzione `selectionSort` (riga 29), quindi stampa di nuovo array dopo che è stato ordinato.

```

1 // fig13_01.c
2 // Algoritmo di ordinamento per selezione.
3 #define SIZE 10
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <time.h>
7
8 // prototipi di funzioni
9 void selectionSort(int array[], size_t length);
10 void swap(int array[], size_t first, size_t second);
11 void printPass(int array[], size_t length, int pass, size_t index);
12
13 int main(void) {
14     int array[SIZE] = {0}; // dichiara l'array di int da ordinare
15
16     srand(time(NULL)); // fornisce il seme alla funzione rand
17
18     for (size_t i = 0; i < SIZE; i++) {
19         array[i] = rand() % 90 + 10; // assegna un valore a ogni elemento
20     }
21
22     puts("Unsorted array:");
23
24     for (size_t i = 0; i < SIZE; i++) { // stampa l'array

```

```
25     printf("%d ", array[i]);
26 }
27
28 puts("\n");
29 selectionSort(array, SIZE);
30 puts("Sorted array:");
31
32 for (size_t i = 0; i < SIZE; i++) { // stampa l'array
33     printf("%d ", array[i]);
34 }
35
36 puts("");
37 }
38
39 // funzione che ordina per selezione l'array
40 void selectionSort(int array[], size_t length) {
41     // esegui l'iterazione su length - 1 elementi
42     for (size_t i = 0; i < length - 1; i++) {
43         size_t smallest = i; // primo indice dell'array rimanente
44
45         // trova l'indice dell'elemento piu' piccolo
46         for (size_t j = i + 1; j < length; j++) {
47             if (array[j] < array[smallest]) {
48                 smallest = j;
49             }
50         }
51
52         swap(array, i, smallest); // scambia con l'elemento piu' piccolo
53         printPass(array, length, i + 1, smallest); // stampa il passo
54     }
55 }
56
57 // funzione che scambia due elementi nell'array
58 void swap(int array[], size_t first, size_t second) {
59     int temp = array[first];
60     array[first] = array[second];
61     array[second] = temp;
62 }
63
64 // funzione che stampa un passo dell'algoritmo
65 void printPass(int array[], size_t length, int pass, size_t index) {
66     printf("After pass %2d: ", pass);
67
68     // stampa gli elementi fino all'elemento selezionato
69     for (size_t i = 0; i < index; i++) {
70         printf("%d ", array[i]);
71     }
72
73     printf("%d* ", array[index]); // indica l'elemento scambiato
74
75     // completa la stampa dell'array
76     for (size_t i = index + 1; i < length; i++) {
77         printf("%d ", array[i]);
```

```

78     }
79
80     printf("%s", "\n"); // per l'allineamento
81
82     // indica la porzione dell'array che e' ordinata
83     for (int i = 0; i < pass; i++) {
84         printf("%s", "--");
85     }
86
87     puts(""); // aggiungi un newline
88 }

```

```

Unsorted array:
12 34 88 14 32 12 34* 77 56 83
After pass 1: 12 34 88 14 32 72* 34 77 56 83
After pass 2: 12 14 88 34* 32 72 34 77 56 83
After pass 3: 12 14 32 34 88* 72 34 77 56 83
After pass 4: 12 14 32 34* 88 72 34 77 56 83
After pass 5: 12 14 32 34 34 72 88* 77 56 83
After pass 6: 12 14 32 34 34 56 88 77 72* 83
After pass 7: 12 14 32 34 34 56 72 77 88* 83
After pass 8: 12 14 32 34 34 56 72 77* 88 83
After pass 9: 12 14 32 34 34 56 72 77 83 88*
Sorted array:
12 14 32 34 34 56 72 77 83 88

```

Figura 13.1 Algoritmo di ordinamento per selezione.

Nella funzione `selectionSort` (righe 40-55), la variabile `smallest` (riga 43) memorizza l'indice dell'elemento restante più piccolo. Le righe 42-54 iterano per `length - 1` volte. La riga 43 assegna a `smallest` l'indice `i`, il primo indice della porzione non ordinata dell'array. Le righe 46-50 elaborano i restanti elementi. Per ciascun elemento, la riga 47 verifica se il valore dell'elemento è minore del valore di quello all'indice `smallest`. Se è minore, la riga 48 assegna l'indice dell'elemento corrente a `smallest`. Terminato il ciclo, `smallest` contiene l'indice dell'elemento restante più piccolo. La riga 52 chiama la funzione `swap` (righe 58-62) per scambiare i valori nelle posizioni `j` e `smallest`, inserendo l'elemento restante più piccolo alla posizione `i` nell'array.

Nell'output vengono usati trattini per indicare la porzione dell'array che è ordinata dopo ogni passo dell'algoritmo sull'array. Abbiamo posto un asterisco accanto alla posizione dell'elemento che è stato scambiato con l'elemento più piccolo in quel passo. L'elemento alla sinistra dell'asterisco e l'elemento sopra i trattini più a destra sono i due valori che sono stati scambiati a ogni passo.

13.3.2 Efficienza dell'ordinamento per selezione

L'algoritmo di ordinamento per selezione viene eseguito in un tempo $O(n^2)$. Nella nostra funzione `selectionSort`, il ciclo esterno (righe 42-54) elabora i primi $n - 1$ elementi dell'array, inserendo l'elemento restante più piccolo nella sua posizione ordinata. Il ciclo interno (righe 46-50) elabora gli elementi rimanenti, cercando quello più piccolo. Questo ciclo viene eseguito $n - 1$ volte durante la prima iterazione del ciclo esterno, $n - 2$ volte durante la seconda iterazione, poi $n - 3, \dots, 3, 2, 1$. Quindi, questo ciclo interno effettua un totale di $n(n - 1)/2$ o $(n^2 - n)/2$ iterazioni. Nella notazione O grande, i termini più piccoli e le costanti vengono ignorati, lasciando un valore $O(n^2)$.

✓ Autovalutazione

1. *(Discussione)* Il seguente array riflette il risultato dopo il primo passo di un ordinamento per selezione:

4 56 34 10 77 51 93 30 5 52

Che cosa fa il secondo passo? Mostrate l'array modificato.

Risposta: Il secondo passo scambia 56 con 5 (il secondo elemento più piccolo), producendo:

4 5 34 10 77 51 93 30 56 52

2. *(Codice)* Esaminate la seguente funzione `selectionSort`:

```

1 void selectionSort(int array[], size_t length) {
2     // esegui l'iterazione su length - 1 elementi
3     for (size_t i = 0; i < length - 1; i++) {
4         size_t smallest = i; // primo indice dell'array rimanente
5
6         // trova l'indice dell'elemento più' piccolo
7         for (size_t j = i + 1; j < length; j++) {
8             if (array[j] < array[smallest]) {
9                 ???
10            }
11        }
12
13        swap(array, i, smallest); // scambia con l'elemento più' piccolo
14        printPass(array, length, i + 1, smallest); // stampa il passo
15    }
16 }
```

Quale istruzione deve essere inserita al posto di ??? alla riga 9 per completare il codice?

Risposta: `smallest = j;`

13.4 Ordinamento per inserzione

L'ordinamento per inserzione (*insertion sort*) è un altro algoritmo di ordinamento semplice ma inefficiente. La prima iterazione di questo algoritmo prende il secondo elemento dell'array e, se è minore del primo elemento, allora li scambia. La seconda iterazione considera il terzo elemento e lo inserisce nella posizione corretta rispetto ai primi due elementi, così tutti e tre gli elementi sono ordinati. Alla i -esima iterazione di questo algoritmo, vengono ordinati i primi i elementi dell'array originario.

Considerate come esempio il seguente array:

34 56 4 10 77 51 93 30 5 52

La prima iterazione dell'algoritmo di ordinamento per inserzione guarda i primi due elementi dell'array, 34 e 56. Questi due elementi sono già ordinati, quindi l'algoritmo continua. Se non fossero ordinati, l'algoritmo li scambierebbe.

All'iterazione successiva l'algoritmo considera il terzo valore, 4. Questo valore è minore di 56, perciò l'algoritmo memorizza 4 in una variabile temporanea e sposta 56 di una posizione a destra. L'algoritmo quindi verifica che 4 è minore di 34 e così sposta 34 di una posizione a destra. L'algoritmo ha ora raggiunto l'inizio dell'array, per cui mette 4 nella posizione 0, producendo

4	34	56	10	77	51	93	30	5	52
---	----	----	----	----	----	----	----	---	----

All'iterazione successiva l'algoritmo memorizza il valore 10 in una variabile temporanea. Poi l'algoritmo confronta 10 con 56 e sposta 56 di una posizione a destra perché è più grande di 10. L'algoritmo quindi confronta 10 con 34, spostando 34 a destra di una posizione. Quando l'algoritmo confronta 10 con 4, osserva che 10 è più grande di 4 e mette 10 nella posizione 1, producendo

4	10	34	56	77	51	93	30	5	52
---	----	----	----	----	----	----	----	---	----

Alla i -esima iterazione, i primi $i + 1$ elementi dell'array originario saranno ordinati l'uno rispetto all'altro, ma non è detto che siano nella loro posizione finale, perché ci potrebbero essere valori più piccoli posizionati nel resto dell'array.

13.4.1 Implementazione dell'ordinamento per inserzione

Il programma della Figura 13.2 implementa l'algoritmo di ordinamento per inserzione. Nella funzione `insertionSort` (righe 39-55), la variabile `insert` (riga 43) contiene l'elemento da inserire finché non vengono spostati gli altri elementi. Le righe 41-54 elaborano gli elementi dell'array dall'indice 1 fino alla fine. Ogni iterazione memorizza in `moveItem` (riga 42) la posizione in cui verrà inserito l'elemento e memorizza in `insert` (riga 43) il valore che verrà inserito nella porzione ordinata dell'array. Le righe 46-50 individuano la posizione in cui va inserito l'elemento. Il ciclo termina sia quando il programma raggiunge l'inizio dell'array sia quando esso raggiunge un elemento che è minore del valore da inserire. La riga 48 sposta un elemento verso destra, e la riga 49 decrementa la posizione nella quale inserire l'elemento seguente. Al termine del ciclo annidato, la riga 52 inserisce l'elemento in posizione. L'output di questo programma usa trattini per indicare la porzione dell'array che è ordinata dopo ogni passo. Un asterisco è posto accanto all'elemento che viene inserito nella posizione ordinata in quel passo.

```

1 // fig13_02.c
2 // Algoritmo di ordinamento per inserzione.
3 #define SIZE 10
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <time.h>
7
8 // prototipi di funzioni
9 void insertionSort(int array[], size_t length);
10 void printPass(int array[], size_t length, int pass, size_t index);
11
12 int main(void)
13     int array[SIZE] = {0}; // dichiara l'array di int da ordinare
14
15     srand(time(NULL)); // fornisce il seme alla funzione rand
16
17     for (size_t i = 0; i < SIZE; i++) {
18         array[i] = rand() % 90 + 10; // assegna un valore a ogni elemento
19     }
20
21     puts("Unsorted array:");
22
23     for (size_t i = 0; i < SIZE; i++) { // stampa l'array
24         printf("%d ", array[i]);

```

```
25 }
26 puts("\n");
27 insertionSort(array, SIZE);
28 puts("Sorted array:");
29
30 for (size_t i = 0; i < SIZE; i++) { // stampa l'array
31     printf("%d ", array[i]);
32 }
33
34 puts("");
35 }
36 }
37
38 // funzione che ordina l'array
39 void insertionSort(int array[], size_t length) {
40     // esegui l'iterazione su length - 1 elementi
41     for (size_t i = 1; i < length; i++) {
42         size_t moveItem = i; // posizione in cui inserire l'elemento
43         int insert = array[i]; // contiene l'elemento da inserire
44
45         // cerca la posizione dove mettere l'elemento corrente
46         while (moveItem > 0 && array[moveItem - 1] > insert) {
47             // sposta l'elemento a destra di una posizione
48             array[moveItem] = array[moveItem - 1];
49             --moveItem;
50         }
51
52         array[moveItem] = insert; // inserisci l'elemento al suo posto
53         printPass(array, length, i, moveItem);
54     }
55 }
56
57 // funzione che stampa un passo dell'algoritmo
58 void printPass(int array[], size_t length, int pass, size_t index) {
59     printf("After pass %2d: ", pass);
60
61     // stampa gli elementi fino all'elemento selezionato
62     for (size_t i = 0; i < index; i++) {
63         printf("%d ", array[i]);
64     }
65
66     printf("%d* ", array[index]); // indica l'elemento inserito
67
68     // termina di stampare l'array
69     for (size_t i = index + 1; i < length; i++) {
70         printf("%d ", array[i]);
71     }
72
73     printf("%s", "\n"); // per l'allineamento
74
75     // indica la porzione di array che e' ordinata
76     for (size_t i = 0; i <= pass; i++) {
77         printf("%s", "-- ");
```

```

78     }
79
80     puts(""); // aggiungi un newline
81 }

```

```

Unsorted array:
72 16 11 92 63 99 59 82 99 30

After pass 1: 16* 72 11 92 63 99 59 82 99 30

After pass 2: 11* 16 72 92 63 99 59 82 99 30

After pass 3: 11 16* 72 92* 63 99 59 82 99 30

After pass 4: 11 16 63* 72 92 99 59 82 99 30

After pass 5: 11 16 63 72 92 99* 59 82 99 30

After pass 6: 11 16 59* 63 72 92 99 82 99 30

After pass 7: 11 16 59 63 72 82* 92 99 99 30

After pass 8: 11 16 59 63 72 82 92 99 99* 30

After pass 9: 11 16 30* 59 63 72 82 92 99 99

Sorted array:
11 16 30 59 63 72 82 92 99 99

```

Figura 13.2 Algoritmo di ordinamento per inserzione.

13.4.2 Efficienza dell'ordinamento per inserzione

L'algoritmo di ordinamento per inserzione, come quello di ordinamento per selezione, viene eseguito in un tempo $O(n^2)$. Come nella funzione `selectionSort` nel Paragrafo 13.3.1, la funzione `insertionSort` usa cicli annidati. Il ciclo esterno (righe 41-54) effettua l'iterazione `SIZE - 1` volte, inserendo a ogni iterazione un elemento nella posizione corretta fra gli elementi ordinati fino a quel punto. Per gli scopi di questa applicazione, `SIZE - 1` è equivalente a $n - 1$, poiché `SIZE` è il numero di elementi dell'array. Il ciclo annidato (righe 46-50) effettua l'iterazione sugli elementi precedenti nell'array. Nel caso peggiore, questo ciclo `while` richiede $n - 1$ confronti. Ogni singolo ciclo viene eseguito in un tempo $O(n)$. Nella notazione O grande, in caso di cicli annidati si devono moltiplicare i numeri di iterazioni per ogni ciclo. Per ogni iterazione di un ciclo esterno ci sarà un certo numero di iterazioni del ciclo interno. In questo algoritmo, per ognuna delle $O(n)$ iterazioni del ciclo esterno vi saranno $O(n)$ iterazioni del ciclo interno. Moltiplicare questi valori produce un valore O grande pari a $O(n^2)$.

✓ Autovalutazione

1. (*Completare*) La prima iterazione dell'algoritmo per inserzione prende in considerazione il secondo elemento dell'array e, se è minore del primo, lo scambia con esso. La seconda iterazione considera il terzo elemento e lo inserisce nella posizione corretta rispetto ai primi due elementi, così tutti e tre gli elementi sono ordinati. Dopo la i -esima iterazione di questo algoritmo, vengono ordinati i _____ elementi dell'array.

Risposta: primi i .

2. (*Discussione*) Entrambi gli algoritmi di ordinamento per inserzione e di ordinamento per selezione vengono eseguiti in un tempo $O(n^2)$. Qual è la struttura del programma, presente in entrambi i casi, che determina il tempo di esecuzione $O(n^2)$?

Risposta: Ciascun algoritmo ha un ciclo `for` annidato.

13.5 Caso pratico: visualizzazione dell'algoritmo di ordinamento per fusione ad alte prestazioni

L'ordinamento per fusione (*merge sort*) è un algoritmo di ordinamento efficiente ma concettualmente più complesso di quelli per selezione e per inserzione. L'algoritmo di ordinamento per fusione ordina un array suddividendolo in due sottoarray della stessa dimensione, ordinando ognuno di essi e fondendoli infine in un array più grande. Con un numero dispari di elementi, l'algoritmo crea i due sottoarray, di cui uno con un elemento in più dell'altro.

L'implementazione dell'ordinamento per fusione in questo esempio è ricorsiva. Il caso di base è dato da un array con un solo elemento che è, per sua natura, già ordinato. Quindi, l'ordinamento termina immediatamente quando viene chiamato con un array di un elemento. Il passo di ricorsione suddivide un array di due o più elementi in due sottoarray della stessa dimensione, li ordina ricorsivamente, quindi li fonde in un array ordinato più grande. Ancora, se c'è un numero dispari di elementi, un sottoarray è di un elemento più grande dell'altro.

Supponete che l'algoritmo abbia già fuso piccoli array per creare gli array ordinati A:

4 10 34 56 77

e B:

5 30 51 52 93

L'ordinamento per fusione combina questi due array in uno ordinato più grande. L'elemento più piccolo in A è 4 (posizionato nell'elemento 0). L'elemento più piccolo in B è 5 (posizionato nell'elemento 0). Per determinare l'elemento più piccolo nell'array più grande, l'algoritmo confronta 4 e 5. Il valore in A è più piccolo, e così 4 diventa il primo elemento nell'array combinato. Quindi l'algoritmo confronta 10 (elemento 1 in A) con 5 (elemento 0 in B). Il valore in B è più piccolo e così 5 diventa il secondo elemento nell'array più grande. L'algoritmo continua confrontando 10 con 30, con 10 che diventa il terzo elemento nell'array, e così via.

13.5.1 Implementazione dell'ordinamento per fusione

Il programma della Figura 13.3 implementa l'algoritmo di ordinamento per fusione. Le righe 35-37 definiscono la funzione `mergeSort`. La riga 36 chiama la funzione `sortSubArray` con 0 e `length - 1` come argomenti (`length` è la dimensione dell'array). Gli argomenti corrispondono agli indici iniziali e finali dell'array da ordinare, facendo sì che `sortSubArray` operi sull'intero array. La funzione `sortSubArray` è definita nelle righe 40-62. La riga 42 verifica il caso di base. Se la dimensione del sottoarray è 1, il sottoarray è ordinato, per cui la funzione termina immediatamente. Se la dimensione del sottoarray è maggiore di 1, la funzione suddivide il sottoarray in due, chiama ricorsivamente la funzione `sortSubArray` per ordinare le due metà e poi le unisce. La riga 56 chiama ricorsivamente la funzione `sortSubArray` sulla prima metà del sottoarray e la riga 57 chiama ricorsivamente la funzione `sortSubArray` sulla seconda metà. Quando viene restituito il risultato di queste due chiamate a funzione, ognuna delle due metà è ordinata. La riga 60 chiama la funzione `merge` (righe 65-111) sulle due metà per unirle in un sottoarray ordinato più grande.

```

1 // fig13_03.c
2 // Algoritmo di ordinamento per fusione.
3 #define SIZE 10
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <time.h>
7
8 // prototipi di funzioni

```

```
9 void mergeSort(int array[], size_t length);
10 void sortSubArray(int array[], size_t low, size_t high);
11 void merge(int array[], size_t left, size_t middle1,
12           size_t middle2, size_t right);
13 void displayElements(int array[], size_t length);
14 void displaySubArray(int array[], size_t left, size_t right);
15
16 int main(void) {
17     int array[SIZE] = {0}; // dichiara l'array di int da ordinare
18
19     srand(time(NULL)); // fornisci il seme alla funzione rand
20
21     for (size_t i = 0; i < SIZE; i++) {
22         array[i] = rand() % 90 + 10; // assegna un valore a ogni elemento
23     }
24
25     puts("Unsorted array:");
26     displayElements(array, SIZE); // stampa l'array
27     puts("\n");
28     mergeSort(array, SIZE); // ordina per fusione l'array
29     puts("Sorted array:");
30     displayElements(array, SIZE); // stampa l'array
31     puts("");
32 }
33
34 // funzione che ordina per fusione l'array
35 void mergeSort(int array[], size_t length) {
36     sortSubArray(array, 0, length - 1);
37 }
38
39 // funzione che ordina una porzione dell'array
40 void sortSubArray(int array[], size_t low, size_t high) {
41     // effettua il test per il caso di base: la dimensione dell'array è 1
42     if ((high - low) >= 1) { // se non si tratta del caso di base...
43         size_t middle1 = (low + high) / 2;
44         size_t middle2 = middle1 + 1;
45
46         // stampa il passo di suddivisione
47         printf("%s", "split:   ");
48         displaySubArray(array, low, high);
49         printf("%s", "\n      ");
50         displaySubArray(array, low, middle1);
51         printf("%s", "\n      ");
52         displaySubArray(array, middle2, high);
53         puts("\n");
54
55         // dividi l'array a metà e ordina ciascuna metà ricorsivamente
56         sortSubArray(array, low, middle1); // prima metà
57         sortSubArray(array, middle2, high); // seconda metà
58
59         // fondi i due array ordinati
60         merge(array, low, middle1, middle2, high);
61     }
```

```
62 }
63
64 // fonda i due sottoarray ordinati in un sottoarray ordinato
65 void merge(int array[], size_t left, size_t middle1,
66             size_t middle2, size_t right) {
67     size_t leftIndex = left; // indice nel sottoarray sinistro
68     size_t rightIndex = middle2; // indice nel sottoarray destro
69     size_t combinedIndex = left; // indice nell'array temporaneo
70     int tempArray[SIZE] = {0}; // array temporaneo
71
72     // stampa i due sotrray prima di fonderli
73     printf("%s", "merge: ");
74     displaySubArray(array, left, middle1);
75     printf("\n");
76     displaySubArray(array, middle2, right);
77     puts("");
78
79     // fonda i sotrray finche' non si raggiunge la fine di uno di loro
80     while (leftIndex <= middle1 && rightIndex <= right) {
81         // inserisci il piu' piccolo dei due elementi correnti nel risultato
82         // e spostati nella posizione seguente nel sottoarray
83         if (array[leftIndex] <= array[rightIndex]) {
84             tempArray[combinedIndex++] = array[leftIndex++];
85         }
86         else {
87             tempArray[combinedIndex++] = array[rightIndex++];
88         }
89     }
90
91     if (leftIndex == middle2) { // fine del sottoarray sinistro?
92         while (rightIndex <= right) { // copia il sottoarray destro
93             tempArray[combinedIndex++] = array[rightIndex++];
94         }
95     }
96     else { // fine del sottoarray destro?
97         while (leftIndex <= middle1) { // copia il sottoarray sinistro
98             tempArray[combinedIndex++] = array[leftIndex++];
99         }
100    }
101
102    // copia di nuovo i valori nell'array originario
103    for (size_t i = left; i <= right; i++) {
104        array[i] = tempArray[i];
105    }
106
107    // stampa il sottoarray combinato
108    printf("%s", "      ");
109    displaySubArray(array, left, right);
110    puts("\n");
111 }
112
113 // stampa gli elementi dell'array
114 void displayElements(int array[], size_t length) {
```

```
115     displaySubArray(array, 0, length - 1);
116 }
117
118 // stampa alcuni elementi dell'array
119 void displaySubArray(int array[], size_t left, size_t right) {
120     // stampa spazi per l'allineamento
121     for (size_t i = 0; i < left; i++) {
122         printf("%s", "    ");
123     }
124
125     // stampa la porzione di array rimanente
126     for (size_t i = left; i <= right; i++) {
127         printf(" %d", array[i]);
128     }
129 }
```

```

split:      71 44 88 58 23
            71 44 88
                  58 23

split:      71 44 88
            71 44
                  88

split:      71 44
            71
                  44

merge:      71
            44
            44 71

merge:      44 71
            88
            44 71 88

split:      58 23
            58
                  23

merge:      58
            23
            23 58

merge:      44 71 88
            23 58
            23 44 58 71 88

merge:      60 76 79 79 86
            23 44 58 71 88
            23 44 58 60 71 76 79 79 86 88

Sorted array:
23 44 58 60 71 76 79 79 86 88

```

Figura 13.3 Algoritmo di ordinamento per fusione.

Le righe 80-89 nella funzione `merge` eseguono un ciclo fino a giungere alla fine di uno o dell'altro sottoarray. La riga 83 verifica quale elemento all'inizio dei due sottoarray è più piccolo. Se l'elemento nel sottoarray sinistro è più piccolo, la riga 84 lo mette in posizione nell'array combinato. Se l'elemento nel sottoarray destro è più piccolo, la riga 87 lo mette in posizione nell'array combinato. Quando il ciclo `while` viene completato, un intero sottoarray è già inserito nell'array combinato, ma l'altro sottoarray contiene ancora dati. La riga 91 verifica se il sottoarray sinistro è giunto alla fine. Se è così, le righe 92-94 riempiono l'array combinato con gli elementi rimanenti del sottoarray destro. Altrimenti, abbiamo raggiunto la fine del sottoarray destro, e le righe 97-99 riempiono l'array combinato con gli elementi rimanenti del sottoarray sinistro. Infine, le righe 103-105 copiano i valori di `tempArray` nella corretta porzione dell'array originario. L'output di questo programma stampa le suddivisioni e le fusioni eseguite dall'ordinamento per fusione, mostrando il progredire dell'ordinamento a ogni passo dell'algoritmo.

13.5.2 Efficienza dell'ordinamento per fusione

L'ordinamento per fusione è un algoritmo di gran lunga più efficiente sia dell'ordinamento per inserzione che dell'ordinamento per selezione (sebbene ciò possa essere difficile da credere guardando la Figura 13.3, piuttosto elaborata). Considerate la prima chiamata (non ricorsiva) alla funzione `sortSubArray`. Questa produce:

- due chiamate ricorsive alla stessa funzione `sortSubArray`, ognuna con un sottoarray approssimativamente della metà dell'array originario;
- una singola chiamata alla funzione `merge`.

Questa chiamata alla funzione `merge` richiede nel peggio dei casi $n - 1$ confronti per riempire l'array originario, con valore $O(n)$. Ricordate che ogni elemento dell'array viene scelto confrontando un elemento da ognuno dei sottoarray. Le due chiamate alla funzione `sortSubArray` producono:

- altre quattro chiamate ricorsive alla funzione `sortSubArray`, ognuna con un sottoarray di circa un quarto dell'array originario;
- altre due chiamate alla funzione `merge`.

Queste due chiamate alla funzione `merge` richiedono ognuna nel peggio dei casi $n/2 - 1$ confronti, per un numero totale di confronti $O(n)$. Questo processo continua, con ogni chiamata a `sortSubArray` che genera due ulteriori chiamate a `sortSubArray` e una chiamata a `merge`, finché l'algoritmo non suddivide l'array in sottoarray di un solo elemento. A ogni livello sono necessari $O(n)$ confronti per fondere i sottoarray. Ogni livello suddivide a metà gli array, per cui raddoppiare la dimensione dell'array richiede in più solo un ulteriore livello. Quadruplicare la dimensione dell'array richiede due ulteriori livelli. Questo schema è logaritmico e produce $\log_2 n$ livelli. Ciò produce un'efficienza totale pari a $O(n \log n)$.

Esercizi di supporto

Questo caso pratico sulla visualizzazione dell'algoritmo di ordinamento per fusione è integrato da esercizi su altri algoritmi di ordinamento complessi: l'Esercizio 13.6 (bucket sort) e l'Esercizio 13.7 (quicksort). L'algoritmo di bucket sort raggiunge alte prestazioni attraverso un uso intelligente di parecchia più memoria rispetto agli altri ordinamenti che esaminiamo: è un esempio di **compromesso spazio-tempo**.

13.5.3 Riepilogo di valori della notazione O grande di alcuni algoritmi

La tabella seguente riepiloga i valori della notazione O grande degli algoritmi di ordinamento trattati sin qui e per l'algoritmo quicksort, che verrà implementato nell'Esercizio 13.7.

Algoritmo	O grande
Ordinamento per inserzione	$O(n^2)$
Ordinamento per selezione	$O(n^2)$
Ordinamento per fusione	$O(n \log n)$
Bubble sort	$O(n^2)$
Quicksort	Caso peggiore: $O(n^2)$ Caso medio: $O(n \log n)$

La tabella seguente elenca i valori di O grande trattati in questo capitolo riferiti a un certo numero di valori di n per mettere in evidenza le differenze nelle velocità di crescita. La tabella include $O(\log n)$, ovvero la notazione O grande per la ricerca binaria trattata nel Capitolo 6.

n	Valore decimale approssimato	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$
2^{10}	1.000	10	2^{10}	10×2^{10}	2^{20}
2^{20}	1.000.000	20	2^{20}	20×2^{20}	2^{40}
2^{30}	1.000.000.000	30	2^{30}	30×2^{30}	2^{60}

✓ Autovalutazione

1. (*Discussione*) L'algoritmo ricorsivo di ordinamento per fusione ordina un array suddividendolo in due sottoarray della stessa dimensione, ordinando ogni sottoarray e fondendo i sottoarray in un array più grande. I sottoarray nell'ordinamento per fusione non vengono ordinati con gli algoritmi di ordinamento che abbiamo trattato, quali il bubble sort, l'ordinamento per selezione o l'ordinamento per inserzione. Spiegate come vengono ordinati in realtà i sottoarray nell'ordinamento per fusione.

Risposta: Se l'array ha un numero pari di elementi, l'algoritmo di ordinamento per fusione suddivide l'array in due metà della stessa dimensione. Se l'array ha un numero dispari di elementi, una delle due "metà" ha un elemento in più rispetto all'altra. Ciascuna metà viene quindi suddivisa ricorsivamente in due metà più piccole. Il processo di dimezzamento continua fino a quando ciascuna parte contiene un unico elemento. Questo è il caso di base della ricorsione, perché viene ordinato un array di un solo elemento. In seguito, questi elementi individuali vengono fusi in un sottoarray ordinato di due elementi in base ai loro valori. Quindi, in risposta alla domanda, possiamo dire che *l'ordinamento è in realtà semplice*. Durante lo svolgimento della ricorsione dell'algoritmo, l'ordinamento per fusione continua a fondere sottoarray ordinati più piccoli per formare sottoarray ordinati più grandi. Ogni fusione di due sottoarray ordinati risulta in un sottoarray ordinato più grande, di dimensione all'incirca doppia rispetto ai due sottoarray che vengono fusi. L'ultima fusione dà come risultato la versione ordinata dell'array originario.

2. (*Discussione*) Riportiamo qui di seguito le prime tre righe dell'output dell'ordinamento per fusione che abbiamo usato come esempio:

```
split:    79 86 60 79 76 71 44 88 58 23
          79 86 60 79 76
                      71 44 88 58 23
```

e le ultime tre righe dell'output:

```
merge:    60 76 79 79 86
          23 44 58 71 88
          23 44 58 60 71 76 79 79 86 88
```

Confrontate questi output, esprimendo osservazioni coerenti con il modo in cui funziona l'ordinamento per fusione.

Risposta: 1. Nella *fase di fusione finale*, ogni sottoarray corrisponde a uno dei sottoarray non ordinati prodotto dal *primo passo di suddivisione* dell'algoritmo. 2. Ciascun sottoarray che entra nella fase di fusione finale è *ordinato*. 3. L'*array combinato finale*, composto da 10 elementi, contiene lo stesso numero di elementi dell'array non ordinato originario su cui è stato eseguito il primo passo di suddivisione. 4. L'*array combinato finale*, composto da 10 elementi, è di fatto *ordinato*. L'algoritmo ricorsivo di ordinamento per fusione suddivide l'array originario non ordinato in parti sempre più piccole fino ad arrivare a parti composte da un solo elemento, per poi fonderle formando le più piccole parti ordinate dell'array originario. L'algoritmo continua a fondere queste parti ordinate formando altre parti ordinate via via più grandi, fino a fondere le due metà – ora ordinate – dell'array originario non ordinato e formare l'array finale ordinato.

13.6 Riepilogo

Paragrafo 13.1 Introduzione

- L'ordinamento riguarda la disposizione di dati in ordine.

Paragrafo 13.2 Efficienza degli algoritmi: O grande

- Un modo per descrivere l'efficienza di un algoritmo è l'uso della **notazione O grande** (O), la quale indica lo sforzo che un algoritmo può dover compiere per risolvere un problema.
- Per gli algoritmi di ricerca e di ordinamento, la notazione O grande descrive come varia l'**entità dello sforzo** che un particolare algoritmo può dover compiere a seconda di quanti elementi sono presenti nei dati.
- Un algoritmo che è **$O(1)$** si dice che ha un **tempo di esecuzione costante**. Ciò non significa che l'algoritmo richieda un solo confronto. Significa solo che il numero di confronti non cresce quando aumenta la dimensione dell'array.
- Un algoritmo **$O(n)$** ha un **tempo di esecuzione lineare**.
- Un algoritmo **$O(n^2)$** ha **tempo di esecuzione quadratico**.
- La notazione O grande è stata ideata per mettere in evidenza i fattori dominanti, ignorando i termini che diventano insignificanti per valori grandi di n .
- La notazione O grande si riferisce alla velocità di crescita del tempo di esecuzione dell'algoritmo, pertanto le costanti vengono ignorate.

Paragrafo 13.3 Ordinamento per selezione

- La prima iterazione di un **ordinamento per selezione** seleziona l'elemento più piccolo nell'array e lo scambia con il primo elemento. La seconda iterazione seleziona il secondo elemento più piccolo (che è il più piccolo di quelli rimasti) e lo scambia con il secondo elemento. L'algoritmo continua finché l'ultima iterazione seleziona il secondo elemento più grande e lo scambia col penultimo, lasciando l'elemento più grande come ultimo. Alla i -esima iterazione, gli i elementi più piccoli dell'intero array sono ordinati nelle prime i posizioni dell'array.
- L'algoritmo di ordinamento per selezione viene eseguito in un tempo $O(n^2)$.

Paragrafo 13.4 Ordinamento per inserzione

- La prima iterazione di un **ordinamento per inserzione** considera il secondo elemento dell'array e, se è minore del primo, lo scambia con esso. La seconda iterazione considera il terzo elemento e lo inserisce nella posizione corretta rispetto ai primi due elementi. Dopo la i -esima iterazione, i primi i elementi dell'array originario sono ordinati. Sono richieste solo $n - 1$ iterazioni.
- L'algoritmo di ordinamento per inserzione viene eseguito in un tempo $O(n^2)$.

Paragrafo 13.5 Caso pratico: visualizzazione dell'algoritmo di ordinamento per fusione ad alte prestazioni

- L'**algoritmo di ordinamento per fusione** è più veloce ma più complesso da implementare degli ordinamenti per selezione e per inserzione.
- L'algoritmo di ordinamento per fusione ordina un array **suddividendolo** in due **sottoarray** della stessa dimensione, ordinando ogni sottoarray e **fondendo** i sottoarray in un array più grande.
- Il caso di base di un ordinamento per fusione è un array con un elemento, il quale è già ordinato, per cui l'ordinamento per fusione termina immediatamente quando viene chiamato con un array di un elemento. La componente di fusione dell'ordinamento per fusione prende due array ordinati (questi potrebbero essere array di un elemento) e li combina in un array ordinato più grande.

- L'ordinamento per fusione attua la fusione guardando il primo elemento in ogni array, che è anche l'elemento più piccolo in ognuno di essi. L'algoritmo prende il più piccolo di questi due e lo mette nel primo elemento dell'array più grande ordinato. Se vi sono ancora elementi in un sottoarray, l'algoritmo guarda il secondo elemento in quel sottoarray (che è ora il più piccolo fra gli elementi rimasti) e lo confronta con il primo elemento nell'altro sottoarray. L'ordinamento per fusione continua questo processo fino a quando non sono stati elaborati tutti gli elementi in uno dei sottoarray, e quindi aggiunge gli elementi rimasti dell'altro sottoarray all'array più grande.
- L'operazione di fusione dell'algoritmo di ordinamento per fusione viene compiuta su due sottoarray ordinati, ognuno approssimativamente della dimensione $n/2$. Ognuno di questi sottoarray effettua a sua volta $n/2-1$ confronti, ovvero $O(n)$ confronti in totale. Questo schema continua man mano che ogni livello opera su un numero doppio di array, ma ognuno della metà dell'array precedente.
- Questo dimezzamento produce $\log n$ livelli, con ogni livello che richiede $O(n)$ confronti, per un'efficienza totale di $O(n \log n)$, che è di gran lunga più efficiente di $O(n^2)$.

Esercizi di autovalutazione

13.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- Un'applicazione dell'ordinamento per selezione richiederebbe approssimativamente _____ volte il tempo richiesto per un array di 32 elementi per operare su un array di 128 elementi.
- L'efficienza dell'ordinamento per fusione è _____.

13.2 Il valore O grande della ricerca lineare è $O(n)$ e della ricerca binaria è $O(\log n)$. Quale aspetto chiave sia della ricerca binaria (Capitolo 6) sia dell'ordinamento per fusione giustifica la porzione logaritmica dei loro rispettivi valori O grande?

13.3 Per quale aspetto l'ordinamento per inserzione è superiore all'ordinamento per fusione? In che senso l'ordinamento per fusione è superiore all'ordinamento per inserzione?

13.4 Nel testo diciamo che dopo che l'ordinamento per fusione suddivide l'array in due sottoarray, poi ordina questi due sottoarray e li fonde. Perché qualcuno potrebbe essere confuso dalla nostra affermazione che “esso poi ordina questi due sottoarray”?

Risposte agli esercizi di autovalutazione

13.1 a) 16, perché un algoritmo $O(n^2)$ richiede un tempo 16 volte maggiore se il numero di elementi viene quadruplicato. b) $O(n \log n)$.

13.2 Entrambi questi algoritmi incorporano il “dimezzamento”, riducendo in qualche modo qualcosa della metà a ogni passo. La ricerca binaria non prende più in considerazione una metà dell'array dopo ogni confronto. L'ordinamento per fusione suddivide l'array a metà a ogni chiamata ricorsiva.

13.3 L'ordinamento per inserzione è più facile da capire e da implementare dell'ordinamento per fusione. L'ordinamento per fusione è di gran lunga più efficiente – $O(n \log n)$ – dell'ordinamento per inserzione – $O(n^2)$.

13.4 In un certo senso, esso non ordina realmente questi due sottoarray, ma semplicemente continua a suddividere a metà l'array originario finché non arriva ad avere un sottoarray di un elemento, che è, per sua natura, già ordinato. Mette poi insieme i sottoarray originari, fondendo questi array di un elemento per formare sottoarray più grandi, i quali sono poi fusi, e così via.

Esercizi

13.5 (*Ordinamento per selezione ricorsivo*) Un ordinamento per selezione esamina un array alla ricerca del suo elemento più piccolo. Quando l'elemento più piccolo viene trovato, è scambiato con il primo elemento dell'array. Il processo è poi ripetuto per il sottoarray che inizia con il secondo elemento. Ogni iterazione sull'array produce un elemento da mettere nella sua giusta posizione. Questo ordinamento richiede capacità di elabo-

razione simili a quelle del bubble sort. Per un array di n elementi, si devono fare $n - 1$ iterazioni e per ogni sottoarray si devono fare $n - 1$ confronti per trovare il valore più piccolo. Quando il sottoarray da elaborare contiene un elemento, l'array è ordinato. Scrivete una funzione ricorsiva `selectionSort` per eseguire questo algoritmo.

13.6 (Bucket sort) L'algoritmo di bucket sort inizia con un array unidimensionale di interi positivi da ordinare e un array bidimensionale di interi con righe indicizzate da 0 a 9 e colonne indicizzate da 0 a $n - 1$, dove n è il numero dei valori nell'array da ordinare. Ogni riga dell'array con doppio indice è detta bucket (letteralmente "cesta"). Scrivete una funzione `bucketSort` che riceva come argomenti un array intero e la dimensione dell'array.

L'algoritmo è il seguente.

- Eseguite un ciclo lungo l'array unidimensionale e mettete ognuno dei suoi valori in una riga dell'array bidimensionale di bucket basandovi sulla sua cifra delle unità. Per esempio, 97 viene messo nella riga 7, 3 viene messo nella riga 3 e 100 nella riga 0.
- Eseguite un ciclo ordinato lungo le righe e le colonne dell'array di bucket e copiate i valori nell'array originario. Il nuovo ordine dei tre valori considerati sopra nell'array con singolo indice è 100, 3 e 97.
- Ripetete questo processo per ogni successiva posizione di cifra (decine, centinaia, migliaia, e così via) e fermatevi quando è stata elaborata la cifra più a sinistra del numero più grande.

Alla seconda iterazione sull'array, 100 è posizionato nella riga 0, 3 nella riga 0 (avendo una sola cifra, lo trattiamo come 03) e 97 nella riga 9. L'ordine dei valori nell'array con singolo indice è ora 100, 3 e 97. Alla terza iterazione, 100 è posizionato nella riga 1, 3 (003) nella riga 0 e 97 (097) nella riga zero (dopo il 3). Il bucket sort ha tutti i valori correttamente ordinati dopo l'elaborazione della cifra più a sinistra del numero più grande. Esso sa di avere terminato quando tutti i valori sono copiati nella riga zero dell'array con doppio indice.

L'array bidimensionale dei bucket è 10 volte più grande della dimensione dell'array intero da ordinare. Questa tecnica di ordinamento fornisce prestazioni di gran lunga migliori, per esempio, del bubble sort, ma richiede molta più memoria. Il bubble sort richiede solo una locazione di memoria aggiuntiva per il tipo di dati da ordinare. Il bucket sort è un esempio di compromesso spazio-tempo. Esso usa più memoria ma offre prestazioni migliori.

L'algoritmo di bucket sort sopra descritto richiede che vengano copiati di nuovo tutti i dati nell'array originario a ogni passo. Un'altra possibilità è quella di creare un secondo array di bucket bidimensionale e di spostare ripetutamente i dati tra i due array di bucket finché sono tutti copiati nella riga zero di uno degli array. La riga zero contiene alla fine l'array ordinato.

13.7 (Quicksort) Abbiamo discusso diverse tecniche di ordinamento negli esempi e negli esercizi del Capitolo 6 e di questo capitolo. Presentiamo ora la tecnica di ordinamento ricorsivo chiamata quicksort. L'algoritmo di base per un array unidimensionale di valori è il seguente.

- Passo di partizionamento:* prendete il primo elemento dell'array non ordinato e determinate la sua posizione finale nell'array ordinato (ovvero, tutti i valori alla sua sinistra saranno inferiori e tutti i valori alla sua destra saranno superiori). Otteniamo così un elemento nella sua posizione corretta e due sottoarray non ordinati.
- Passo ricorsivo:* eseguite il *passo di partizionamento* su ogni sottoarray non ordinato.

Per ogni sottoarray, il *passo di partizionamento* colloca un altro elemento nella sua posizione finale dell'array ordinato e crea due ulteriori sottoarray non ordinati. Quando un sottoarray è composto da un solo elemento è già ordinato, di conseguenza quell'elemento è nella sua posizione finale.

L'algoritmo di base sembra abbastanza semplice, ma come determiniamo la posizione finale del primo elemento di ogni sottoarray? Come esempio, considerate il seguente insieme di valori (l'elemento in grassetto è l'elemento di partizionamento, che verrà posto nella sua posizione finale nell'array ordinato):

37 2 6 4 89 8 10 12 68 45

- Partendo dall'elemento più a destra dell'array, confrontate ogni elemento con 37 finché non viene trovato un elemento minore di 37. Quindi scambiate 37 con l'altro elemento. Il primo elemento minore di 37 è 12, e così vengono scambiati 37 e 12. Nel nuovo array aggiornato l'elemento 12 è evidenziato in corsivo per indicare che è stato appena scambiato con 37:

12 2 6 4 89 8 10 **37** 68 45

- b) Partendo dalla sinistra dell'array, ma iniziando con l'elemento *dopo* il 12, confrontate ogni elemento con 37 finché non viene trovato un elemento maggiore di 37. Allora scambiate 37 con quell'elemento. Il primo elemento maggiore di 37 è 89, e così vengono scambiati 37 e 89. Il nuovo array è

12 2 6 4 37 8 10 89 68 45

- c) Partendo da destra, ma iniziando con l'elemento *prima* dell'89, confrontate ogni elemento con 37 finché non viene trovato un elemento minore di 37. Allora scambiate 37 con quell'elemento. Il primo elemento minore di 37 è 10, e così vengono scambiati 37 e 10. Il nuovo array è

12 2 6 4 10 8 37 89 68 45

- d) Partendo da sinistra, ma iniziando con l'elemento *dopo* il 10, confrontate ogni elemento con 37 finché non viene trovato un elemento maggiore di 37. Allora scambiate 37 con quell'elemento. Non vi sono più altri elementi maggiori di 37 prima di esso, pertanto quando confrontiamo 37 con se stesso, sappiamo che 37 è stato posto nella sua posizione finale nell'array ordinato.

Una volta che l'array è stato partizionato, vi sono due sottoarray non ordinati. Il sottoarray con valori minori di 37 contiene 12, 2, 6, 4, 10 e 8. Il sottoarray con valori maggiori di 37 contiene 89, 68 e 45. Il quicksort continua partizionando entrambi gli array alla stessa maniera dell'array originario.

Scrivete la funzione ricorsiva `quicksort` per ordinare un array unidimensionale di interi. La funzione deve ricevere come argomenti un array di interi, un indice di inizio e un indice di fine. La funzione `partition` deve essere chiamata da `quicksort` per compiere il passo di partizionamento.

CAPITOLO

14

Sommario del capitolo

- 14.1 Introduzione
- 14.2 Direttiva per il preprocessore `#include`
- 14.3 Direttiva per il preprocessore `#define`: costanti simboliche
- 14.4 Direttiva per il preprocessore `#define`: macro
- 14.5 Compilazione condizionale
- 14.6 Direttive per il preprocessore `#error` e `#pragma`
- 14.7 Operatori `#` e `##`
- 14.8 Numeri di riga
- 14.9 Costanti simboliche predefinite
- 14.10 Afferzioni
- 14.11 Programmazione sicura in C
- 14.12 Riepilogo

Preprocessore

Obiettivi

- Usare `#include` per gestire i file in programmi di grandi dimensioni.
- Usare `#define` per creare macro con e senza argomenti.
- Usare la compilazione condizionale per specificare porzioni di un programma che non sempre devono essere compilate (come il codice di supporto per il debugging).
- Stampare messaggi di errore durante la compilazione condizionale.
- Usare asserzioni per verificare se i valori di alcune espressioni sono corretti.

14.1 Introduzione

L'esecuzione del preprocessore del C avviene *prima* della compilazione di un programma. Esso:

- include altri file nel file da compilare;
- definisce **costanti simboliche e macro**;
- **compila in modo condizionale** il codice del programma;
- **esegue in modo condizionale le direttive per il preprocessore**.

Le direttive per il preprocessore iniziano con `#`. Solo caratteri di spazatura e commenti delimitati da `/*` e `*/` possono comparire su una riga prima di una direttiva per il preprocessore.

Il C ha forse la più grande collezione installata di codice “legacy” (letteralmente “ereditato”) rispetto a qualunque altro moderno linguaggio di programmazione. È usato attivamente da circa cinquant’anni. In qualità di programmatore professionista in C, è probabile che vi imbattiate in codice scritto molti anni fa con l’uso di tecniche di programmazione ormai vecchie. In questo capitolo esamineremo alcune di quelle tecniche, consigliandovene altre più nuove in grado di sostituirle.

✓ Autovalutazione

1. *(Scelta multipla)* Quali delle seguenti azioni sono eseguite dal preprocessore?
 - a) Inclusione di altri file nel file da compilare.
 - b) Definizione di costanti simboliche e macro.
 - c) Compilazione condizionale di codice di programma ed esecuzione condizionale di direttive per il preprocessore.
 - d) Tutte le precedenti.

Risposta: d.

2. *(Vero/Falso)* Il C ha forse la più grande collezione installata di codice “legacy” rispetto a qualunque altro moderno linguaggio di programmazione. È usato attivamente da circa cinquant’anni.

Risposta: Vero.

14.2 Direttiva per il preprocessore #include

In questo testo si è fatto uso della **direttiva per il preprocessore #include**. Quando il preprocessore incontra una direttiva `#include`, la sostituisce con una copia del file specificato. Le due forme della direttiva `#include` sono:

```
#include <filename>
#include "filename"
```

La differenza tra esse sta nella posizione da cui il preprocessore comincia la ricerca del file. Se il nome del file è racchiuso tra parentesi angolari (`< e >`) – come i **file di intestazione della Libreria Standard** – il preprocessore cerca nelle cartelle del compilatore e di sistema in modo *dipendente dall’implementazione*. Solitamente si utilizzano nomi di file racchiusi tra virgolette (`" "`) per includere i file di intestazione che vengono definiti per essere utilizzati con il programma. In questo caso, il preprocessore inizia la ricerca nella *stessa* cartella del file nel quale compare la direttiva `#include`. Se il compilatore non riesce a trovare il file specificato nella cartella corrente, la ricerca proseguirà nelle cartelle del compilatore e di sistema in modo dipendente dall’implementazione.

Oltre che per i file di intestazione della Libreria Standard, la direttiva `#include` è usata frequentemente in programmi costituiti da *più file sorgente*. Vi capiterà spesso di creare file di intestazione per dichiarazioni comuni di un programma, quindi di includere quel file in più file sorgente. Esempi di tali dichiarazioni sono:

- dichiarazioni di strutture (`struct`) e unioni (`union`);
- `typedef`;
- enumerazioni (`enum`);
- prototipi di funzioni.

✓ Autovalutazione

1. *(Completare)* La direttiva per il preprocessore _____ fa sì che una copia del file specificato sia inclusa al posto della direttiva.

Risposta: `#include`.

2. *(Completare)* Se il nome del file di una direttiva `#include` è racchiuso tra virgolette, il preprocessore inizia la sua ricerca del file nella _____.

Risposta: stessa directory del file che viene compilato.

14.3 Direttiva per il preprocessore #define: costanti simboliche

La direttiva `#define` crea:

- *costanti simboliche* (costanti rappresentate come identificatori);
- **macro** (operazioni definite come simboli).

Il formato della direttiva `#define` è:

```
#define identificatore testo-di-sostituzione
```

Per convenzione, l'*identificatore* di una costante simbolica dovrebbe contenere solo lettere maiuscole e trattini bassi (*underscore*). L'utilizzo di nomi significativi per le costanti simboliche aiuta a rendere i programmi autodocumentanti.

Sostituzione di costanti simboliche

Quando il preprocessore incontra una direttiva `#define`, sostituisce l'*identificatore* con il *testo di sostituzione* in tutto il file sorgente, ignorando eventuali occorrenze dell'*identificatore* nei letterali stringa o nei commenti. Per esempio,

```
#define PI 3.14159
```

sostituisce tutte le occorrenze successive della costante simbolica PI con 3.14159. Le costanti simboliche consentono di creare costanti con nome e di usare i loro nomi nel programma.

Errore comune con costanti simboliche

Ogni cosa a destra del nome di una costante simbolica sostituisce la costante simbolica. Per esempio,

```
#define PI = 3.14159
```

fa sì che il preprocessore sostituisca *ogni* occorrenza di PI con "= 3.14159". Le direttive `#define` non corrette causano molti errori logici e di sintassi subdoli. Quindi, rispetto alla precedente direttiva `#define`, potrebbe essere preferibile l'uso di variabili `const`, come:

```
const double PI = 3.14159;
```

Queste hanno l'ulteriore vantaggio di essere definite in C, quindi il compilatore può verificarne la correttezza della sintassi e della sicurezza del tipo.

✓ Autovalutazione

1. (Completare) La direttiva `#define` crea costanti _____ e _____.

Risposta: simboliche, macro.

2. (Vero/Falso) L'istruzione

```
#define PI 3.14159;
```

sostituisce tutte le occorrenze successive della costante simbolica PI con la costante numerica 3.14159.

Risposta: Falso. In realtà, l'esempio di questa direttiva `#define` mostra un errore comune. Le direttive per il preprocessore non sono istruzioni del C e in genere non devono finire con un punto e virgola. L'istruzione riportata sopra sostituirebbe tutte le occorrenze di PI con 3.14159; (includendo il punto e virgola), con buone probabilità di causare uno o più errori di compilazione.

14.4 Direttiva per il preprocessore `#define: macro`

Tecnicamente, qualsiasi identificatore definito in una direttiva per il preprocessore `#define` è una macro. Come avviene con le costanti simboliche, l'identificatore della macro è sostituito con il testo di sostituzione prima che il programma sia compilato. Le macro possono essere definite con o senza argomenti. Una macro senza argomenti è una costante simbolica. Quando il preprocessore incontra una macro con argomenti, sostituisce gli argomenti nel testo di sostituzione, quindi espande la macro, cioè sostituisce la macro con il suo testo di sostituzione e la lista degli argomenti.

14.4.1 Macro con un argomento

Considerate la seguente definizione di macro con un argomento per il calcolo dell'area di un cerchio:

```
#define CIRCLE_AREA(x) ((PI) * (x) * (x))
```

Espandere una macro con un argomento

Dovunque nel file compaia `CIRCLE_AREA(argomento)`, il preprocessore:

- sostituisce l'*argomento* al posto di *x* nel testo di sostituzione;
- sostituisce PI con il suo valore 3.14159 (dal Paragrafo 14.3);
- espande la macro nel programma.

Per esempio, il preprocessore espande

```
double area = CIRCLE_AREA(4);
```

in

```
double area = ((3.14159) * (4) * (4));
```

Al momento della compilazione, il compilatore valuta la precedente espressione e assegna il risultato alla variabile *area*.

Importanza delle parentesi

Le parentesi attorno a ogni *x* nel testo di sostituzione forzano l'ordine giusto di calcolo quando l'argomento della macro è un'espressione. Per esempio, l'istruzione

```
double area = CIRCLE_AREA(c + 2);
```

che viene espansa in

```
double area = ((3.14159) * (c + 2) * (c + 2));
```

viene valutata correttamente perché le parentesi forzano l'ordine giusto di calcolo. Se le parentesi nella definizione della macro sono omesse, l'espansione della macro risulta essere

```
double area = 3.14159 * c + 2 * c + 2;
```

che viene calcolata in modo scorretto come

```
double area = (3.14159 * c) + (2 * c) + 2;
```

a causa delle regole di precedenza degli operatori del C. Per questo motivo, dovreste sempre racchiudere tra parentesi gli argomenti di una macro nel testo di sostituzione per evitare errori logici.

È meglio usare una funzione

Definire la macro `CIRCLE_AREA` come una funzione è più sicuro. La funzione `circleArea`

```
double circleArea(double x) {
    return 3.14159 * x * x;
}
```

esegue lo stesso calcolo della macro CIRCLE_AREA, ma l'argomento della funzione viene calcolato una sola volta quando la funzione viene chiamata. Inoltre, il compilatore esegue il controllo dei tipi sulle funzioni. Il preprocessore non supporta il controllo dei tipi. In passato, i programmatori usavano spesso le macro per sostituire le chiamate di funzioni con un codice in linea, così da eliminare il sovraccarico delle chiamate di funzioni. Gli odierni compilatori ottimizzanti spesso sostituiscono per voi il codice in linea per le chiamate di funzioni, per cui molti programmatori non usano più le macro a questo scopo. Potete usare anche la parola chiave `inline` del C standard (vedi Appendice C).

14.4.2 Macro con due argomenti

Quella che segue è una definizione di macro con due argomenti per il calcolo dell'area di un rettangolo:

```
#define RECTANGLE_AREA(x, y) ((x) * (y))
```

Dovunque `RECTANGLE_AREA(x, y)` compaia nel programma, il preprocessore sostituisce i valori di `x` e `y` nel testo di sostituzione della macro ed espande la macro nel programma. Per esempio, l'istruzione

```
int rectangleArea = RECTANGLE_AREA(a + 4, b + 7);
```

viene espansa in

```
int rectangleArea = ((a + 4) * (b + 7));
```

14.4.3 Carattere di continuazione per macro

Il testo di sostituzione per una macro o una costante simbolica è qualsiasi cosa a destra dell'identificatore nella direttiva `#define`. Se il testo di sostituzione è più lungo del resto della riga, si può mettere un carattere di continuazione `\` alla fine della riga per continuare il testo di sostituzione sulla riga successiva.

14.4.4 Direttiva per il preprocessore `#undef`

Le costanti simboliche e le macro possono essere ignorate nella parte restante di un file sorgente usando la direttriva per il preprocessore `#undef`. La direttriva `#undef` cancella la definizione di una costante simbolica o del nome di una macro. Il campo d'azione di una costante simbolica o di una macro va dalla sua definizione fino alla sua cancellazione con `#undef`, o fino alla fine del file sorgente. Una volta priva di definizione, una macro o una costante simbolica può essere ridefinita con `#define`.

14.4.5 Macro della Libreria Standard

Alcune funzioni della Libreria Standard sono in realtà definite come macro, sulla base di altre funzioni della libreria. Una macro definita comunemente nel file di intestazione `<stdio.h>` è

```
#define getchar() getc(stdin)
```

La definizione della macro `getchar` usa la funzione `getc` per ottenere un carattere dallo stream standard input. Anche la funzione `putchar` del file di intestazione `<stdio.h>` e le funzioni che trattano caratteri del file di intestazione `<cctype.h>` sono spesso implementate come macro.

14.4.6 Evitare espressioni con effetti secondari nelle macro

Le espressioni con *effetti secondari* (per esempio, i valori delle variabili sono modificati) *non* vanno passate a una macro, perché gli argomenti di una macro possono essere valutati più di una volta. Mostreremo un esempio di ciò nel Paragrafo 14.11.

✓ Autovalutazione

1. (*Vero/Falso*) La seguente macro con due argomenti calcola l'area di un rettangolo:

```
#define RECTANGLE_AREA(x, y) ((x) * (y))
```

Ovunque nel programma appaia RECTANGLE_AREA(x, y), i valori di x e y vengono sostituiti nel testo di sostituzione della macro e la macro viene espansa al posto del suo nome. Per esempio, l'istruzione

```
double rectArea = RECTANGLE_AREA(a + 4, b + 7);
```

viene espansa in

```
double rectArea = (a + 4 * b + 7);
```

Il valore dell'espressione viene valutato al momento dell'esecuzione e assegnato alla variabile rectArea.

Risposta: Falso. In realtà, la corretta espansione è:

```
double rectArea = ((a + 4) * (b + 7));
```

2. (*Completere*) Le espressioni con _____ non vanno passate a una macro perché gli argomenti di una macro possono essere valutati più di una volta.

Risposta: effetti secondari.

14.5 Compilazione condizionale

Una compilazione condizionale permette di controllare quali direttive per il preprocessore vengono eseguite e se parti del codice in C vengono compilati. Ogni direttiva condizionale per il preprocessore valuta un'espressione intera costante. Le espressioni di cast, le espressioni con sizeof e le costanti di enumerazione *non possono* essere valutate nelle direttive per il preprocessore.

14.5.1 Direttiva per il preprocessore #if...#endif

Il costrutto condizionale del preprocessore è molto simile all'istruzione di selezione if. Il seguente codice per il preprocessore:

```
#if !defined(MY_CONSTANT)
    #define MY_CONSTANT 0
#endif
```

determina se MY_CONSTANT è *definita* – cioè, se MY_CONSTANT è già comparsa in una direttiva #define precedente nel codice sorgente corrente. L'espressione defined(MY_CONSTANT) ha valore 1 (*vero*) se MY_CONSTANT è definita; altrimenti, ha valore 0 (*falso*). Se il risultato è 0, !defined(MY_CONSTANT) ha valore 1, indicando che MY_CONSTANT non era stata definita precedentemente, quindi la direttiva #define viene eseguita. Altrimenti, il preprocessore salta la direttiva #define.

Ogni costrutto #if termina con #endif. Le direttive #ifdef e #ifndef sono abbreviazioni di #if defined(nome) e #if !defined(nome). Potete testare un costrutto condizionale del preprocessore costituito da più parti usando le direttive:

- #elif (l'equivalente di else if in un'istruzione if);
- #else (l'equivalente di else in un'istruzione if).

Le direttive condizionali per il preprocessore sono usate frequentemente per *evitare che i file di intestazione siano inclusi innumerevoli volte nello stesso file sorgente*. Queste direttive vengono inoltre utilizzate frequentemente per attivare e disattivare il codice che rende il software compatibile con una serie di piattaforme.

14.5.2 Commentare porzioni di codice con #if...#endif

Durante lo sviluppo di un programma è spesso utile "commentare" porzioni di codice per evitare che vengano compilate. Se questo codice contiene commenti su più linee, non è possibile usare /* e */ per svolgere questa operazione, perché tali commenti non possono essere annidati. Invece, potete usare il seguente costrutto del preprocessore:

```
#if 0
    codice da non compilare
#endif
```

Per permettere al codice di essere compilato, sostituite lo 0 nel precedente costrutto con 1.

14.5.3 Compilazione condizionale e codice di debugging

La compilazione condizionale si usa talvolta come aiuto per il *debugging*. Per esempio, alcuni programmati usano istruzioni `printf` per stampare valori delle variabili e per confermare il flusso di controllo di un programma. Queste istruzioni `printf` possono essere racchiuse in direttive condizionali per il preprocessore. In questo modo, le istruzioni vengono compilate solo finché il processo di debugging non è completato. Per esempio,

```
#ifdef DEBUG
    printf("Variable x = %d\n", x);
#endif
```

compila l'istruzione `printf` se la costante simbolica `DEBUG` è definita con

```
#define DEBUG
```

prima di `#ifdef DEBUG`. Quando il debugging è completato, eliminate o commentate la direttiva `#define` nel file sorgente, e le istruzioni `printf` inserite a scopo di debugging sono ignorate durante la compilazione. In programmi più grandi può essere opportuno definire diverse costanti simboliche che controllano la compilazione condizionale in sezioni separate del file sorgente.

Molti compilatori permettono di definire e di cancellare la definizione di costanti simboliche come `DEBUG` con un flag del compilatore che potete fornire ogni volta che compilate il codice in modo da non doverlo cambiare. Quando inserite istruzioni `printf` compilate in modo condizionale in posizioni dove il C si aspetta un'istruzione singola (cioè, il corpo di un'istruzione di controllo), assicuratevi che le istruzioni compilate in modo condizionale siano racchiuse tra parentesi graffe (`{}`).

✓ Autovalutazione

1. *(Vero/Falso)* L'istruzione di compilazione condizionale

```
#ifdef DEBUG
    printf("Variable x = %d\n", x);
#endif
```

compila l'istruzione `printf` se la costante simbolica `DEBUG` è definita (`#define DEBUG`) prima di `#ifdef DEBUG`.

Risposta: *Vero*.

2. *(Vero/Falso)* Durante lo sviluppo del programma, spesso è utile “commentare” porzioni di codice per evitare che vengano compilate. Se questo codice contiene commenti su più righe, bisognerebbe usare `/*` e `*/` per eseguire questa operazione.

Risposta: *Falso*. In realtà, se il codice contiene commenti su più righe, non è possibile usare `/*` e `*/` per svolgere questa operazione, poiché tali commenti non possono essere annidati. Invece, potete usare il seguente costrutto del preprocessore:

```
#if 0
    codice da non compilare
#endif
```

14.6 Direttive per il preprocessore #error e #pragma

La direttiva `#error`

`#error sequenza di token`

stampa un messaggio dipendente dall'implementazione che include i *token* specificati nella direttiva. I token sono sequenze di caratteri separati da spazi. Per esempio,

`#error 1 - Out of range error`

contiene 6 token. Quando su alcuni sistemi viene elaborata una direttiva `#error`, i token sono stampati come un messaggio di errore, la preelaborazione si arresta e il programma non viene compilato.

La direttiva `#pragma`

`#pragma sequenza di token`

provoca un'azione *definita dall'implementazione*. Una direttiva `#pragma` non riconosciuta dall'implementazione è ignorata. Per maggiori informazioni su `#error` e `#pragma`, guardate la documentazione relativa al vostro compilatore C.

✓ Autovalutazione

1. (*Completare*) Quando su alcuni sistemi viene elaborata una direttiva _____, i token nella direttiva sono stampati come un messaggio di errore, la preelaborazione si arresta e il programma non viene compilato.

Risposta: `#error`.

2. (*Completare*) La direttiva `#pragma` provoca un'azione _____.

Risposta: definita dall'implementazione.

14.7 Operatori # e

L'operatore `#` fa sì che i token di un testo di sostituzione siano convertiti in stringhe tra virgolette. Considerate la seguente definizione di macro:

```
#define HELLO(x) puts("Hello, " #x);
```

Quando `HELLO(John)` compare nel file di un programma, il preprocessore lo espande in

```
puts("Hello, " "John");
```

sostituendo `#x` con la stringa `"John"`. Le stringhe separate da spaziature sono concatenate durante la preelaborazione, perciò l'istruzione precedente è equivalente a

```
puts("Hello, John");
```

L'operatore `#` deve essere usato in una macro con argomenti poiché l'operando di `#` si riferisce a un argomento della macro.

L'operatore `##` concatena due token. Considerate la seguente definizione di macro:

```
#define TOKENCONCAT(x, y) x ## y
```

Quando in un file compare `TOKENCONCAT`, il preprocessore concatena gli argomenti e usa il risultato per sostituire la macro. Per esempio, `TOKENCONCAT(0, K)` è sostituito da `0K` nel programma. L'operatore `##` deve avere due operandi.

✓ Autovalutazione

1. (*Completare*) L'operatore del preprocessore _____ fa sì che i token di un testo di sostituzione siano convertiti in stringhe tra virgolette.

Risposta: `#`.

2. (*Codice*) L'operatore del preprocessore ## concatena due token. Scrivete la definizione di macro descritta da "Quando SIDEBYSIDE compare nel programma, i suoi argomenti sono concatenati e usati per sostituire la macro. Quindi, SIDEBYSIDE(GOOD, BYE) viene sostituito da GOODBYE nel programma."

Risposta: `#define SIDEBYSIDE(a, b) a ## b`

14.8 Numeri di riga

La direttiva per il preprocessore #line fa sì che le righe successive di un codice sorgente siano rinumerate, iniziando con il valore intero costante specificato. La direttiva

`#line 100`

inizia a numerare le righe da 100 cominciando dalla riga successiva del codice sorgente. L'inclusione di un nome di file nella direttiva #line, come in

`#line 100 "file1.c"`

indica che le righe sono numerate da 100 cominciando dalla riga successiva del codice sorgente, e che il nome di file da usare nei messaggi del compilatore è "file1.c". Questa versione della direttiva #line è normalmente usata per rendere più significativi i messaggi prodotti da errori di sintassi e i messaggi di avvertimento del compilatore. I numeri delle righe non compaiono nel file sorgente.

✓ Autovalutazione

1. (*Vero/Falso*) La direttiva per il preprocessore

`#line 100 "file1.c"`

indica che le righe sono numerate da 100 cominciando dalla riga successiva del codice sorgente, e che il nome di file da usare nei messaggi del compilatore è "file1.c".

Risposta: *Vero*.

14.9 Costanti simboliche predefinite

Il C standard fornisce costanti simboliche predefinite, alcune delle quali sono mostrate nella tabella seguente:

Costante simbolica	Spiegazione
<code>_LINE_</code>	Il numero della riga corrente del codice sorgente (una costante intera).
<code>_FILE_</code>	Il nome del file sorgente (una stringa).
<code>_DATE_</code>	La data in cui il file sorgente è stato compilato (nel formato "Mm dd yyyy", come "Jan 19 2002").
<code>_TIME_</code>	L'ora in cui il file sorgente è stato compilato (nel formato "hh:mm:ss").
<code>_STDC_</code>	Il valore 1 se il compilatore supporta il C standard, altrimenti 0. Richiede il flag del compilatore /Za in Visual C++.

Le restanti costanti simboliche predefinite sono nel Paragrafo 6.10.8 del C standard. Questi identificatori iniziano e terminano con *due* trattini bassi e spesso sono utili per includere informazioni aggiuntive nei messaggi di errore. Questi identificatori e l'identificatore `defined` (usato nel Paragrafo 14.5) non possono essere usati nelle direttive `#define` o `#undef`.

✓ Autovalutazione

1. (*Completare*) La costante simbolica predefinita _____ è descritta da “Il valore 1 se il compilatore supporta il C standard, altrimenti 0”.

Risposta: STDC.

14.10 Asserzioni

La macro `assert` (definita in `<assert.h>`) testa il valore di un'espressione al momento dell'esecuzione. Se il valore è falso (0), `assert` stampa un messaggio di errore e termina il programma chiamando la funzione `abort` della libreria di utilità generali (`<stdlib.h>`).

La macro `assert` è uno strumento di debugging utile per testare la correttezza del valore di una variabile. Per esempio, supponete che in un programma la variabile `x` non debba mai essere più grande di 10. Si può usare un'asserzione per verificare il valore di `x` e stampare un messaggio di errore se è maggiore di 10, come in

```
assert(x <= 10);
```

Se `x` è maggiore di 10 quando viene eseguita questa istruzione, il programma stampa un messaggio di errore contenente il numero della riga e il nome del file in cui appare l'istruzione `assert`, poi termina. Potete quindi concentrarvi su quest'area del codice per trovare l'errore.

Se viene definita la costante simbolica `NDEBUG`, le asserzioni che seguono nel file sorgente sono *ignoreate*. Pertanto, quando le asserzioni non sono più necessarie, anziché eliminare ciascuna asserzione manualmente, è possibile inserire la seguente riga nel file sorgente:

```
#define NDEBUG
```

Molti compilatori hanno modalità di debug e rilascio che definiscono e cancellano la definizione `NDEBUG` automaticamente.

Le asserzioni non vanno intese come un sostituto per il trattamento degli errori durante le normali condizioni nella fase di esecuzione. Il loro uso andrebbe limitato alla ricerca degli errori logici durante lo sviluppo del programma. Il C standard comprende anche una funzionalità chiamata `_Static_assert`, che è essenzialmente una versione di `assert` per la fase di compilazione che produce un *errore in fase di compilazione* se l'asserzione fallisce. Esamineremo `_Static_assert` nell'Appendice C.

✓ Autovalutazione

1. (*Vero/Falso*) La macro `assert` (definita in `<assert.h>`) testa il valore di un'espressione in fase di compilazione.

Risposta: *Falso*. In realtà, la macro `assert` testa il valore di un'espressione durante la *fase di esecuzione*. `_Static_assert` è essenzialmente una versione di `assert` per la fase di compilazione che produce un errore in fase di compilazione se l'asserzione fallisce.

2. (*Completare*) Quando le asserzioni non sono più necessarie, potete inserire la riga _____ nel file del codice anziché eliminare ciascuna asserzione manualmente.

Risposta: `#define NDEBUG`.

14.11 Programmazione sicura in C

La macro `CIRCLE_AREA` definita nel Paragrafo 14.4:

```
#define CIRCLE_AREA(x) ((PI) * (x) * (x))
```

è considerata una macro *poco sicura*, perché valuta il suo argomento `x` più di una volta. Questo causa errori subdoli. Se l'argomento di una macro contiene *effetti secondari* – come l'incremento di una variabile o la chiamata di una funzione che modifica il valore di una variabile – essi verrebbero eseguiti più volte.

Per esempio, se chiamiamo `CIRCLE_AREA` come segue:

```
double result = CIRCLE_AREA(++radius);
```

il preprocessore espande la chiamata in

```
double result = ((3.14159) * (++radius) * (++radius));
```

che incrementa `radius` *due volte*. Inoltre, il risultato dell'istruzione precedente è *indefinito*, perché il C permette a una variabile di essere modificata *una volta sola* in un'istruzione. In una chiamata di una funzione l'argomento viene *valutato una volta sola* prima di essere passato alla funzione. Pertanto, le funzioni sono sempre preferite a macro poco sicure.

✓ Autovalutazione

1. (*Completare*) La macro CIRCLE_AREA

```
#define CIRCLE_AREA(x) ((PI) * (x) * (x))
```

è considerata poco sicura poiché _____. Questo può causare errori subdoli.

Risposta: valuta il suo argomento `x` più di una volta.

2. (*Vero/Falso*) Le macro sono sempre preferite alle funzioni.

Risposta: *Falso*. In realtà, le funzioni sono sempre preferite a macro poco sicure.

14.12 Riepilogo

Paragrafo 14.1 Introduzione

- L'esecuzione del preprocessore del C avviene prima della compilazione di un programma.
- Tutte le direttive per il preprocessore iniziano con #.
- Prima di una direttiva per il preprocessore possono comparire su una riga solo caratteri di spaziatura e commenti.

Paragrafo 14.2 Direttiva per il preprocessore #include

- La direttiva `#include` include una copia del file specificato. Se il nome del file è racchiuso tra virgolette, il preprocessore inizia la ricerca nella stessa cartella del file da compilare. Se il nome del file è racchiuso tra parentesi angolari (< e >), come nel caso dei file di intestazione della Libreria Standard del C, la ricerca viene compiuta in una maniera definita dall'implementazione.

Paragrafo 14.3 Direttiva per il preprocessore #define: costanti simboliche

- La direttiva per il preprocessore `#define` crea costanti simboliche e macro.
- Una costante simbolica è un nome per una costante.

Paragrafo 14.4 Direttiva per il preprocessore #define: macro

- Una **macro** è un'operazione definita in una direttiva per il preprocessore `#define`. Le macro possono essere definite con o senza argomenti.
- Il **testo di sostituzione** viene specificato dopo l'identificatore di una costante simbolica o dopo la parentesi destra di chiusura della lista di argomenti di una macro. Se il testo di sostituzione per una macro o una costante simbolica è più lungo del resto della riga, usate un **backslash** (\) alla fine della riga per indicare che il testo di sostituzione continua sulla riga seguente.
- Le costanti simboliche e le macro possono essere cancellate usando la **direttiva per il preprocessore #undef**. La direttiva `#undef` "cancella la definizione" di una costante simbolica o del nome di una macro.
- Il **campo d'azione** di una costante simbolica o di una macro va dalla sua definizione fino alla sua cancellazione con `#undef` o fino alla fine del file.

Paragrafo 14.5 Compilazione condizionale

- La **compilazione condizionale** consente di controllare l'esecuzione delle direttive per il preprocessore e la compilazione del codice del programma.
- Le **direttive condizionali per il preprocessore** valutano espressioni intere costanti. Le espressioni di cast, le espressioni con `sizeof` e le costanti di enumerazione non possono essere valutate nelle direttive per il preprocessore.
- Ogni costrutto `#if` termina con `#endif`.
- Le direttive `#ifdef` e `#ifndef` possono essere utilizzate come abbreviazioni per `#if defined(nome)` e `#if !defined(nome)`.
- I **costrutti condizionali del preprocessore** costituiti da più parti possono essere testati con direttive `#elif` e `#else`.

Paragrafo 14.6 Direttive per il preprocessore `#error` e `#pragma`

- La **direttiva `#error`** arresta la preelaborazione, impedisce la compilazione e stampa un messaggio dipendente dall'implementazione che include i token specificati nella direttiva.
- La **direttiva `#pragma`** causa un'azione definita dall'implementazione. Se la direttiva `#pragma` non viene riconosciuta dall'implementazione viene ignorata.

Paragrafo 14.7 Operatori `#` e `##`

- L'**operatore `#`** fa sì che i token di un *testo di sostituzione* siano convertiti in stringhe tra virgolette. L'operatore `#` deve essere usato in una macro con argomenti perché l'operando di `#` deve essere un argomento della macro.
- L'**operatore `##`** concatena due token. L'operatore `##` deve avere due operandi.

Paragrafo 14.8 Numeri di riga

- La **direttiva per il preprocessore `#line`** fa sì che le righe successive di un codice sorgente siano rinumerate, iniziando con il valore intero costante specificato. Questa direttiva consente anche di specificare il nome usato per il file di quel codice sorgente nei messaggi di errore del compilatore.

Paragrafo 14.9 Costanti simboliche predefinite

- La costante `__LINE__` è il numero (un intero) della riga corrente del codice sorgente.
- La costante `__FILE__` è il nome del file (una stringa).
- La costante `__DATE__` è la data di compilazione del file sorgente (una stringa).
- La costante `__TIME__` è l'ora di compilazione del file sorgente (una stringa).
- La costante `__STDC__` indica se il compilatore supporta il C standard.
- Ognuna delle costanti simboliche predefinite inizia e termina con due trattini bassi.

Paragrafo 14.10 Asserzioni

- La macro `assert` (file di intestazione `<assert.h>`) verifica il valore di un'espressione. Se il valore è 0 (falso), `assert` stampa un messaggio di errore e chiama la funzione `abort` per terminare l'esecuzione del programma.

Esercizi di autovalutazione

14.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.

- Ogni direttiva per il preprocessore deve iniziare con _____.

- b) Il costrutto di compilazione condizionale può essere esteso per analizzare diversi casi usando le direttive _____ e _____.
- c) La direttiva _____ crea macro e costanti simboliche.
- d) Solo i caratteri _____ possono comparire su una riga prima di una direttiva per il preprocessore.
- e) La direttiva _____ elimina le costanti simboliche e i nomi delle macro.
- f) Le direttive _____ e _____ sono fornite come abbreviazioni per `#if defined(nome)` e `#if !defined(nome)`.
- g) La _____ vi consente di controllare l'esecuzione delle direttive per il preprocessore e la compilazione del codice del programma.
- h) La macro _____ stampa un messaggio e termina l'esecuzione del programma se il valore dell'espressione che la macro calcola è 0.
- i) La direttiva _____ inserisce un file in un altro file.
- j) L'operatore del preprocessore _____ concatena i suoi due argomenti.
- k) L'operatore del preprocessore _____ converte il suo operando in una stringa.
- l) Il carattere _____ indica che il testo di sostituzione per una costante simbolica o una macro continua sulla riga successiva.
- m) La direttiva _____ fa sì che le righe di un codice sorgente siano numerate a partire dal valore indicato iniziando dalla riga di codice sorgente successiva.

14.2 Scrivete un programma per stampare i valori delle costanti simboliche predefinite elencate nel Paragrafo 14.9.

14.3 Scrivete una direttiva per il preprocessore per eseguire ognuna delle seguenti operazioni.

- Definire la costante simbolica YES impostandola al valore 1.
- Definire la costante simbolica NO impostandola al valore 0.
- Includere il file di intestazione `common.h`. Il file di intestazione si trova nella stessa directory del file che viene compilato.
- Rinumerare le restanti righe nel file cominciando con il numero di riga 3000.
- Se la costante simbolica TRUE è definita, cancellare la sua definizione e ridefinirla come 1. Non usare `#ifdef`.
- Se la costante simbolica TRUE è definita, cancellare la sua definizione e ridefinirla come 1. Usare la direttiva per il preprocessore `#ifdef`.
- Se la costante simbolica TRUE non è uguale 0, definire la costante simbolica FALSE come 0. Altrimenti definire FALSE come 1.
- Definire la macro `CUBE_VOLUME` che calcola il volume di un cubo. La macro riceve un argomento.

Risposte agli esercizi di autovalutazione

14.1 a) #. b) `#elif`, `#else`. c) `#define`. d) di spaziatura. e) `#undef`. f) `#ifdef`, `#ifndef`. g) compilazione condizionale. h) `assert`. i) `#include`. j) `##`. k) `\`. l) `\`. m) `#line`.

14.2 Vedi sotto. [Nota: in Visual Studio, `_STDC_` richiede il flag del compilatore `/Za`.]

```

1 // ex14_02.c
2 // Stampa i valori delle macro predefinite
3 #include <stdio.h>
4 int main(void) {
5     printf("__LINE__ = %d\n", __LINE__);
6     printf("__FILE__ = %s\n", __FILE__);
7     printf("__DATE__ = %s\n", __DATE__);
8     printf("__TIME__ = %s\n", __TIME__);
9     printf("__STDC__ = %d\n", __STDC__);
10 }
```

```
__LINE__ = 5
__FILE__ = ex14_02.c
__DATE__ = Jan 01 2021
__TIME__ = 11:39:12
__STDC__ = 1
```

- 14.3 a) `#define YES 1`
 b) `#define NO 0`
 c) `#include "common.h"`
 d) `#line 3000`
 e) `#if defined(TRUE)
 #undef TRUE
 #define TRUE 1
 #endif`
 f) `#ifdef TRUE
 #undef TRUE
 #define TRUE 1
 #endif`
 g) `#if TRUE
 #define FALSE 0
 #else
 #define FALSE 1
 #endif`
 h) `#define CUBE_VOLUME(x) ((x) * (x) * (x))`

Esercizi

- 14.4 (*Volume di una sfera*) Scrivete un programma che definisce una macro con un argomento per calcolare il volume di una sfera. Il programma deve calcolare il volume per sfere con raggio da 1 a 10 e stampare i risultati in formato tabellare. La formula per il volume di una sfera è

$$(4.0 / 3) * \pi * r^3$$

dove π è 3.14159.

- 14.5 (*Sommare due numeri*) Scrivete un programma che definisce la macro SUM con due argomenti, x e y, e usate SUM per produrre il seguente output:

```
The sum of x and y is 13
```

- 14.6 (*Il più piccolo di due numeri*) Scrivete un programma che definisca e usi la macro MINIMUM2 per determinare il più piccolo di due valori numerici.

- 14.7 (*Il più piccolo di tre numeri*) Scrivete un programma che definisca e usi la macro MINIMUM3 per determinare il più piccolo di tre valori numerici. La macro MINIMUM3 deve usare la macro MINIMUM2 definita nell'Esercizio 14.6 per determinare il numero più piccolo.

- 14.8 (*Stampare una stringa*) Scrivete un programma che definisca e usi la macro PRINT per stampare un valore stringa.

- 14.9 (*Stampare un array*) Scrivete un programma che definisca e usi la macro PRINTARRAY per stampare un array di interi. La macro deve ricevere come argomenti l'array e il numero dei suoi elementi.

- 14.10 (*Calcolare il totale dei valori di un array*) Scrivete un programma che definisca e usi la macro SUMARRAY per sommare i valori in un array numerico. La macro deve ricevere come argomenti l'array e il numero dei suoi elementi.

CAPITOLO

15

Sommario del capitolo

- 15.1 Introduzione
- 15.2 Liste di argomenti di lunghezza variabile
- 15.3 Uso degli argomenti della riga di comando
- 15.4 Compilazione di programmi con più file sorgente
- 15.5 Terminazione di programmi con `exit` e `atexit`
- 15.6 Suffissi per letterali interi e in virgola mobile
- 15.7 Gestione dei segnali
- 15.8 Allocazione dinamica della memoria: funzioni `calloc` e `realloc`
- 15.9 `goto`: salto non condizionato
- 15.10 Riepilogo

Altri argomenti

Obiettivi

- Scrivere funzioni che usano liste di argomenti di lunghezza variabile.
- Elaborare gli argomenti della riga di comando.
- Compilare programmi con più file sorgente.
- Assegnare tipi specifici a costanti numeriche.
- Terminare programmi con `exit` e `atexit`.
- Gestire eventi asincroni esterni in un programma.
- Allocare array in maniera dinamica e ridimensionare la memoria che era stata allocata dinamicamente in precedenza.

15.1 Introduzione

Questo capitolo presenta ulteriori argomenti non trattati di solito in corsi introduttivi. Molte delle funzionalità esaminate qui sono specifiche di particolari sistemi operativi, specialmente di macOS/Linux e Windows.

15.2 Liste di argomenti di lunghezza variabile

La maggior parte dei programmi nel testo ha usato la funzione della Libreria Standard `printf`. Come minimo, `printf` deve ricevere una stringa come suo primo argomento, ma `printf` può ricevere qualsiasi numero di argomenti aggiuntivi. Il prototipo di funzione per `printf` è:

```
int printf(const char *format, ...);
```

L'ellissi (...) nel prototipo di funzione indica che la funzione riceve un numero variabile di argomenti di qualsiasi tipo. Potete usare questa sintassi per definire le vostre funzioni con **liste di argomenti di lunghezza variabile**.



L'ellissi deve essere l'ultimo parametro; porla al centro di una lista dei parametri è un errore di sintassi.

La seguente tabella contiene le macro e le definizioni dei file di intestazione per argomenti variabili (<stdarg.h>) per costruire funzioni con liste di argomenti di lunghezza variabile:

Identificatore	Spiegazione
va_list	Un tipo adatto a contenere le informazioni che servono alle macro va_start, va_arg e va_end. Per accedere agli argomenti in una lista di argomenti di lunghezza variabile, deve essere definito un oggetto di tipo va_list.
va_start	Una macro che va invocata prima che si possa accedere agli argomenti di una lista di argomenti di lunghezza variabile. La macro inizializza l'oggetto dichiarato con va_list che viene usato dalle macro va_arg e va_end.
va_arg	Una macro che viene espansa al valore dell'argomento successivo della lista di argomenti di lunghezza variabile. Il valore ha il tipo specificato come secondo argomento della macro. Ogni utilizzo di va_arg modifica l'oggetto dichiarato con va_list facendo sì che esso punti all'argomento successivo nella lista.
va_end	Una macro che facilita un normale ritorno da una funzione, alla cui lista di argomenti di lunghezza variabile faceva riferimento la macro va_start.

La Figura 15.1 mostra una funzione average (righe 23-36) con una lista di argomenti di lunghezza variabile. Il primo argomento della funzione è il numero di valori per cui calcolare la media.

```

1 // fig15_01.c
2 // Uso di liste di argomenti di lunghezza variabile
3 #include <stdarg.h>
4 #include <stdio.h>
5
6 double average(int i, ...); // ... rappresenta argomenti variabili
7
8 int main(void) {
9     double w = 37.5;
10    double x = 22.5;
11    double y = 1.7;
12    double z = 10.2;
13
14    printf("%s%.1f; %s%.1f; %s%.1f; %s%.1f\n\n",
15          "w = ", w, "x = ", x, "y = ", y, "z = ", z);
16    printf("%s%.3f\n%s%.3f\n%s%.3f\n",
17          "The average of w and x is ", average(2, w, x),
18          "The average of w, x, and y is ", average(3, w, x, y),
19          "The average of w, x, y, and z is ", average(4, w, x, y, z));
20 }
21
22 // calcola la media
23 double average(int i, ...) {
24     double total = 0; // inizializza total
25     va_list ap; // memorizza le informazioni per va_start e va_end
26
27     va_start(ap, i); // inizializza l'oggetto di tipo va_list
28
29     // elabora la lista di argomenti di lunghezza variabile
30     for (int j = 1; j <= i; ++j) {
31         total += va_arg(ap, double);

```

```

32     }
33
34     va_end(ap); // pulitura della lista di argomenti di lunghezza variabile
35     return total / i; // media calcolata
36 }

```

```
w = 37.5; x = 22.5; y = 1.7; z = 10.2
```

```
The average of w and x is 30.000
The average of w, x, and y is 20.567
The average of w, x, y, and z is 17.975
```

Figura 15.1 Uso di liste di argomenti di lunghezza variabile.

La funzione `average` (righe 23-36) usa tutte le definizioni e le macro del file di intestazione `<stdarg.h>`, eccetto `va_copy` (Paragrafo C.7.8), che è stato aggiunto nel C11. L'oggetto `ap` (abbreviazione di “*argument pointer*”; riga 25) di tipo `va_list` è usato dalle macro `va_start`, `va_arg` e `va_end` per elaborare la lista di argomenti di lunghezza variabile della funzione `average`. La funzione inizia invocando la macro `va_start` (riga 27) per inizializzare l'oggetto `ap` che viene usato in `va_arg` e `va_end`. La macro `va_start` riceve:

- l'oggetto `ap`;
- l'identificatore dell'argomento più a destra nella lista di argomenti *prima* dell'ellissi, `i` in questo caso (`va_start` usa questo argomento per determinare da dove inizia la lista di argomenti di lunghezza variabile).

Dopo, la funzione `average` somma ripetutamente gli argomenti nella lista di argomenti di lunghezza variabile alla variabile `total` (righe 30-32). La macro `va_arg` recupera il valore successivo da sommare a `total`. La macro riceve due argomenti:

- l'oggetto `ap`;
- il *tipo* del valore atteso nella lista di argomenti (`double` in questo caso).

La macro restituisce il valore dell'argomento. La riga 34 invoca la macro `va_end` con l'oggetto `ap` come argomento per facilitare un normale ritorno alla funzione chiamante da `average`. Infine, la riga 35 calcola la media e la restituisce a `main`.

Ci si potrebbe domandare come le funzioni con liste di argomenti a lunghezza variabile come `printf` e `scanf` sappiano quale tipo usare in ogni chiamata alla macro `va_arg`. La risposta è che, in fase di esecuzione del programma, esse esaminano gli specificatori di conversione nella stringa di controllo del formato per determinare il tipo dell'argomento successivo da elaborare.

✓ Autovalutazione

1. (*Completare*) Il prototipo di funzione per `printf` è

```
int printf(const char *format, ...);
```

L'ellissi (...) del prototipo indica che la funzione riceve _____.

Risposta: un numero variabile di argomenti di qualsiasi tipo.

2. (*Scelta multipla*) Quale macro corrisponde alla descrizione: “Per accedere agli argomenti in una lista di argomenti di lunghezza variabile, deve essere definito un oggetto di questo tipo”.

- a) `va_start`
- b) `va_end`
- c) `va_list`
- d) `va_arg`

Risposta: c.

15.3 Uso degli argomenti della riga di comando

Gli argomenti della riga di comando sono comunemente usati per passare opzioni e nomi di file a un programma. La funzione `main` può ricevere argomenti da una riga di comando se la sua lista dei parametri include `int argc` e `char *argv[]`:

- il parametro `argc` riceve il numero di argomenti della riga di comando che l'utente ha inserito;
- il parametro `argv` è un array di stringhe in cui sono memorizzati gli argomenti della riga di comando.

Il programma della Figura 15.2 copia un file in un altro file un carattere alla volta. Supponiamo che il file eseguibile per il programma sia chiamato `mycopy`. Una tipica riga di comando per l'esecuzione di questo programma è:

```
mycopy input output
```

Questa riga di comando indica che il file `input` va copiato nel file `output`. Quando il programma viene eseguito, se `argc` non ha il valore 3 (`mycopy` stesso conta come uno degli argomenti), il programma stampa un messaggio di errore (riga 8) e termina. Altrimenti, l'array `argv` contiene le stringhe "mycopy", "input" e "output". Il programma usa il secondo e il terzo argomento della riga di comando come nomi di file.

```

1 // fig15_02.c
2 // Uso degli argomenti della riga di comando
3 #include <stdio.h>
4
5 int main(int argc, char *argv[])
6 {
7     // controlla il numero degli argomenti della riga di comando
8     if (argc != 3) {
9         puts("Usage: mycopy infile outfile");
10    }
11    FILE *inFilePtr = NULL; // puntatore al file di input
12
13    // tenta l'apertura del file di input
14    if ((inFilePtr = fopen(argv[1], "r")) != NULL) {
15        FILE *outFilePtr = NULL; // puntatore al file di output
16
17        // tenta l'apertura del file di output
18        if ((outFilePtr = fopen(argv[2], "w")) != NULL) {
19            int c = 0; // contiene i caratteri letti dal file sorgente
20
21            // legge e invia in uscita i caratteri
22            while ((c = fgetc(inFilePtr)) != EOF) {
23                fputc(c, outFilePtr);
24            }
25
26            fclose(outFilePtr); // chiudi il file di output
27        }
28        else { // non e' stato possibile aprire il file di output
29            printf("File \"%s\" could not be opened\n", argv[2]);
30        }
31
32        fclose(inFilePtr); // chiudi il file di input
33    }
34    else { // non e' stato possibile aprire il file di input
35        printf("File \"%s\" could not be opened\n", argv[1]);
36    }
}
```

```

36      }
37  }
38 }
```

Figura 15.2 Uso degli argomenti della riga di comando.

Usiamo la funzione `fopen` per aprire questi file per la lettura (riga 14) e la scrittura (riga 18), rispettivamente. Se il programma apre entrambi i file con successo, le righe 22-24 leggono i caratteri dal file `input` e li scrivono nel file `output`. Questo processo continua fino al raggiungimento della fine del file `input`, poi il programma termina. Il risultato è una copia esatta del file `input` (se non si verificano errori durante l'elaborazione). [Nota: nel Visual C++, gli argomenti della riga di comando vengono specificati facendo clic con il pulsante destro del mouse sul nome del progetto nel Solution Explorer e selezionando **Properties**, poi espandendo **Configuration Properties**, selezionando **Debugging** e inserendo gli argomenti nella casella di testo alla destra di **Command Arguments**.]

✓ Autovalutazione

1. (*Completare*) Potete passare gli argomenti della riga di comando a `main` includendo parametri `int argc` e _____ nella lista dei parametri di `main`.

Risposta: `char *argv[]`.

2. (*Discussione*) Supponendo che `inFilePtr` rappresenti un file di input aperto con successo e `outFilePtr` rappresenti un file di output aperto con successo, cosa fa il seguente segmento di codice?

```

while ((c = fgetc(inFilePtr)) != EOF) {
    fputc(c, outFilePtr);
}
```

Risposta: Questo ciclo legge un carattere alla volta dal file di input e lo scrive sul file di output finché non si incontra l'indicatore di end-of-file del file di input.

15.4 Compilazione di programmi con più file sorgente

È possibile costruire programmi costituiti da più file sorgente. Vi sono diverse cose di cui tener conto quando si creano programmi in più file. Per esempio, la definizione di una funzione deve essere interamente contenuta in un solo file (non può abbracciare due o più file).

15.4.1 Dichiarazioni `extern` per variabili globali in altri file

Nel Capitolo 5 abbiamo introdotto i concetti di classe di memoria e campo d'azione. Abbiamo appreso che le variabili dichiarate *al di fuori* di una definizione di funzione sono *variabili globali*. Le variabili globali sono accessibili a ogni funzione definita nello stesso file dopo la loro dichiarazione. Le variabili globali sono accessibili anche alle funzioni in altri file se vengono dichiarate in ogni file in cui vengono usate. Per esempio, per riferirsi alla variabile intera globale `flag` in un altro file, si può usare la dichiarazione

```
extern int flag;
```

Lo specificatore della classe di memoria `extern` indica che la variabile `flag` viene definita *in seguito nello stesso file o in un file differente*. Il compilatore informa il linker che nel file compaiono riferimenti non risolti alla variabile `flag`. Se il linker trova una definizione globale adeguata, risolve i riferimenti a `flag`. Se il linker non riesce a localizzare una definizione di `flag`, emette un messaggio di errore e non produce un file eseguibile. Qualunque identificatore dichiarato con campo d'azione esteso al file è `extern` per impostazione predefinita. Le variabili globali vanno evitate a meno che non siano critiche le prestazioni dell'applicazione perché violano il principio del privilegio minimo.

15.4.2 Prototipi di funzioni

Proprio come le dichiarazioni `extern` possono essere usate per dichiarare che le variabili globali vengono definite in altri file del programma, i prototipi di funzioni possono estendere il campo d'azione di una funzione oltre il file in cui essa è definita. Lo specificatore `extern` non è necessario in un prototipo di funzione. Includete semplicemente il prototipo della funzione in ogni file in cui la funzione è invocata e compilare i file insieme (vedi Paragrafo 14.2). I prototipi di funzioni indicano che la funzione specificata è definita o in seguito nello stesso file o in un file differente. Anche in questo caso, il compilatore non tenta di risolvere i riferimenti a una tale funzione: questo compito è lasciato al linker. Se il linker non riesce a localizzare un'adeguata definizione della funzione, emette un messaggio di errore.

Come esempio di uso di prototipi di funzioni per estendere il campo d'azione di una funzione, considerate un qualsiasi programma contenente la direttiva per il preprocessore `#include <stdio.h>`, la quale include un file contenente i prototipi di funzioni per funzioni come `printf`, `scanf` e molte altre. Un file con `#include <stdio.h>` può usare `printf` e `scanf`, anche se sono definite in altri file. Non ci serve sapere dove sono definite. Il linker risolve automaticamente i nostri riferimenti a queste funzioni.

Riusabilità del software

 La scrittura di programmi in più file sorgente facilita la riusabilità e una buona ingegneria del software. Le funzioni che sono comuni a molte applicazioni devono essere memorizzate nei loro file sorgente. Ogni file sorgente deve avere un corrispondente file di intestazione contenente i prototipi delle funzioni. Ciò consente ai programmati di applicazioni differenti di riutilizzare lo stesso codice includendo il file di intestazione adatto e compilando le loro applicazioni assieme al file sorgente corrispondente.

15.4.3 Restringere il campo d'azione con `static`

È possibile restringere il campo d'azione di una funzione o di una variabile globale al file in cui essa è definita. L'applicazione dello specificatore della classe di memoria `static` a una funzione o a una variabile globale impedisce che questa venga usata al di fuori del file in cui è definita. Questo è chiamato **collegamento interno**. Le funzioni e le variabili globali non precedute da `static` nelle loro definizioni hanno un **collegamento esterno**. È possibile accedervi in altri file contenenti dichiarazioni appropriate.

La dichiarazione di variabile globale

```
static const double PI = 3.14159;
```

crea la variabile costante `PI` di tipo `double`, la inizializza a `3.14159` e indica che `PI` è nota *solo* alle funzioni nel file in cui è definita.

Lo specificatore `static` si usa comunemente con le funzioni di utilità che vengono chiamate solo dalle funzioni in un particolare file. Se una funzione non è necessaria al di fuori di un particolare file, va fatto valere il principio del privilegio minimo applicando `static` alla definizione e al prototipo della funzione.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- Le variabili dichiarate al di fuori di qualsiasi definizione di funzione sono variabili globali.
- Le variabili globali sono accessibili a qualsiasi funzione definita nello stesso file dopo che la variabile è stata dichiarata.
- Le variabili globali sono accessibili anche alle funzioni di altri file.
- Una volta definita una variabile globale, è nota a tutti i file dell'applicazione.

Risposta: d) è *falsa*. In realtà, le variabili globali devono essere dichiarate in *ogni* file nel quale sono usate.

2. (*Completare*) Qualsiasi identificatore dichiarato nel campo d'azione file è _____ per impostazione predefinita.

Risposta: `extern`.

3. (*Vero/Falso*) Dovreste preferire le variabili globali alle variabili locali perché applicano il principio del privilegio minimo.

Risposta: Falso. In realtà, dovreste evitare le variabili globali a meno che le prestazioni dell'applicazione non siano critiche perché esse *violano* il principio del privilegio minimo.

4. (*Complemare*) L'applicazione di static a una funzione o a una variabile globale impedisce che essa sia usata da una funzione che non è definita nello stesso file; questo è chiamato collegamento _____.

Risposta: interno.

5. (*Vero/Falso*) La seguente definizione di variabile globale indica che PI è nota solo alle funzioni nel file in cui è definito:

```
static const double PI = 3.14159;
```

Risposta: Vero.

15.5 Terminazione di programmi con exit e atexit

La libreria di utilità generali (<stdlib.h>) fornisce metodi per terminare l'esecuzione di programmi con modalità diverse da un ritorno convenzionale dalla funzione main.

Funzione exit

La funzione **exit** provoca la terminazione immediata di un programma. Questa funzione è usata spesso per terminare un programma quando viene scoperto un errore. La funzione riceve un argomento, solitamente EXIT_SUCCESS o EXIT_FAILURE, che contiene valori definiti dall'implementazione per la terminazione riuscita o non riuscita con successo.

Funzione atexit

La funzione **atexit** registra una funzione da chiamare quando il programma termina raggiungendo la fine di main o quando viene invocata **exit**. Questa funzione prende come argomento il nome di un'altra funzione. Ricordiamo che il nome di una funzione è un puntatore a quella funzione. Le funzioni chiamate alla fine del programma non possono avere argomenti e non possono restituire un valore. Quando un programma termina, tutte le funzioni precedentemente registrate con **atexit** vengono richiamate nell'ordine inverso a quello della loro registrazione.

Usare le funzioni exit e atexit

Il programma della Figura 15.3 testa le funzioni **exit** e **atexit**. Il programma richiede all'utente di definire se il programma va terminato con **exit** oppure quando raggiunge la fine di **main**. La funzione **print** viene eseguita in ogni caso al termine del programma.

```

1 // fig15_03.c
2 // Uso delle funzioni exit e atexit
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 void print(void); // prototipo
7
8 int main(void) {
9     atexit(print); // registra la funzione print
10    puts("Enter 1 to terminate program with function exit\n"
11        "Enter 2 to terminate program normally");
12    int answer = 0; // utente
13    scanf("%d", &answer);
14
15    // chiama exit se la risposta è 1
16    if (answer == 1) {
17        puts("\nTerminating program with function exit");

```

```

18     exit(EXIT_SUCCESS);
19 }
20
21 puts("\nTerminating program by reaching the end of main");
22 }
23
24 // stampa il messaggio prima della terminazione
25 void print(void) {
26     puts("Executing function print at program termination\n"
27         "Program terminated");
28 }

```

```

Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
1
Terminating program with function exit
Executing function print at program termination
Program terminated

Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
2
Terminating program by reaching the end of main
Executing function print at program termination
Program terminated

```

Figura 15.3 Uso delle funzioni `exit` e `atexit`.

✓ Autovalutazione

1. (*Vero/Falso*) La funzione `atexit` termina un programma immediatamente.

Risposta: *Falso*. In realtà, è la funzione `exit` a causare la terminazione immediata di un programma. La funzione `atexit` registra una funzione da chiamare quando un programma termina raggiungendo la fine di `main` o quando viene invocata `exit`.

2. (*Vero/Falso*) Chiamando `exit` con `EXIT_SUCCESS` viene restituito 1 all'ambiente chiamante, e chiamando `exit` con `EXIT_FAILURE` viene restituito 0.

Risposta: *Falso*. In realtà, chiamando `exit` con `EXIT_SUCCESS` o `EXIT_FAILURE` vengono restituiti *valori definiti dall'implementazione* per una terminazione riuscita o non riuscita con successo.

15.6 Suffixi per letterali interi e in virgola mobile

I suffissi per valori interi e in virgola mobile consentono di specificare esplicitamente i tipi di dati di valori letterali. Come impostazione predefinita, il tipo di un letterale intero viene determinato dal primo tipo in grado di memorizzare il valore: `int`, poi `long int`, poi `unsigned long int` ecc. Un letterale in virgola mobile senza suffisso è di tipo `double`.

I suffissi interi sono `u` o `U` per un `unsigned int`, `l` o `L` per un `long int`, e `ll` o `LL` per un `long long int`. `L` e `LL` sono preferiti per leggibilità, in quanto la lettera minuscola `l` può essere confusa con un `1` (uno). È possibile combinare `u` o `U` con i suffissi per `long int` e `long long int` per creare letterali senza segno per i tipi

interi più grandi. I seguenti letterali sono di tipo `unsigned int`, `long int`, `unsigned long int` e `unsigned long long int`:

```
174u
8358L
28373ul
9876543210llu
```

I suffissi per letterali in virgola mobile sono `f` o `F` per un `float`, e `l` o `L` per un `long double`. Anche in questo caso, è preferibile usare `L` per la leggibilità. I seguenti sono letterali `float` e `long double`:

```
1.28f
3.14159L
```

✓ Autovalutazione

1. (*Completare*) Il C fornisce _____ per valori interi e in virgola mobile per specificare esplicitamente i tipi di dati di valori letterali interi e in virgola mobile.

Risposta: suffissi.

2. (*Completare*) Le seguenti costanti hanno tipi _____ e _____.

```
1.28f
3.14159L
```

Risposta: `float`, `long double`.

15.7 Gestione dei segnali

Un **evento**, o **segnaletico**, esterno asincrono può far sì che un programma termini prematuramente. Alcuni eventi includono:

- interruzioni, come la digitazione `<Ctrl> c` (Linux o Windows) o `<Comando> c` (macOS);
- ordini di terminazione da parte del sistema operativo.

La libreria per la gestione dei segnali (`<signal.h>`) dà la possibilità ai programmi di **intervallare** eventi inaspettati tramite la funzione `signal`, che riceve due argomenti:

- un numero di segnale intero;
- un puntatore a una funzione per la gestione del segnale.

I segnali possono essere generati con la funzione `raise`, la quale riceve come argomento un numero di segnale intero. La tabella seguente riepiloga i segnali standard definiti nel file di intestazione `<signal.h>`.

Segnale	Spiegazione
<code>SIGABRT</code>	Terminazione anomala del programma (es. una chiamata alla funzione <code>abort</code>).
<code>SIGFPE</code>	Un'operazione aritmetica erronea, come una divisione per zero o un'operazione che genera un overflow.
<code>SIGILL</code>	Rilevazione di un'istruzione non permessa.
<code>SIGINT</code>	Ricezione di un segnale di attenzione interattivo (<code><Ctrl> c</code> o <code><Comando> c</code>).
<code>SIGSEGV</code>	Un tentativo di accedere alla memoria che non è allocata per un programma.
<code>SIGTERM</code>	Una richiesta di terminazione inviata al programma.

Dimostrazione della gestione del segnale

Il programma della Figura 15.4 usa la funzione `signal` per *intercettare* un SIGINT. La riga 11 chiama `signal` con SIGINT e un puntatore alla funzione `signalHandler`. Quando si rileva un segnale di tipo SIGINT, il controllo passa alla funzione `signalHandler`, che stampa un messaggio e dà all'utente l'opzione di continuare l'esecuzione normale del programma. Se l'utente desidera continuare l'esecuzione, la riga 49 reinizializza il gestore (*handler*) del segnale chiamando di nuovo `signal`, e il controllo ritorna al punto nel programma in cui il segnale era stato rilevato.

```
1 // fig15_04.c
2 // Uso della gestione dei segnali
3 #include <signal.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <time.h>
7
8 void signalHandler(int signalValue); // prototipo
9
10 int main(void) {
11     signal(SIGINT, signalHandler); // registra il gestore del segnale
12     srand(time(NULL));
13
14     // stampa i numeri da 1 a 100
15     for (int i = 1; i <= 100; ++i) {
16         int x = 1 + rand() % 50; // numero a caso per emettere un SIGINT
17
18         // genera un SIGINT quando x e' 25
19         if (x == 25) {
20             raise(SIGINT);
21         }
22
23         printf("%d", i);
24
25         // stampa \n quando i e' un multiplo di 10
26         if (i % 10 == 0) {
27             printf("\n");
28         }
29     }
30 }
31
32 // gestisce il segnale
33 void signalHandler(int signalValue) {
34     printf("\n%d\n", signalValue);
35     "Interrupt signal (", signalValue, ") received.\n";
36     "Do you wish to continue (1 = yes or 2 = no)? ";
37     int response = 0; // utente
38     scanf("%d", &response);
39
40     // controlla la correttezza della risposta
41     while (response != 1 && response != 2) {
42         printf("(1 = yes or 2 = no)? ");
43         scanf("%d", &response);
44     }
45 }
```

```

46 // determina se continuare
47 if (response == 1) {
48     // riregistra il gestore del segnale per il SIGINT successivo
49     signal(SIGINT, signalHandler);
50 }
51 else {
52     exit(EXIT_SUCCESS);
53 }
54 }
```

```

1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36

Interrupt signal (2) received.
Do you wish to continue (1 = yes or 2 = no)? 1
37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92

Interrupt signal (2) received.
Do you wish to continue (1 = yes or 2 = no)? 2
```

Figura 15.4 Uso della gestione dei segnali.

In questo programma, la funzione `raise` simula un SIGINT. Si sceglie un numero a caso tra 1 e 50. Se il numero è 25, la riga 20 chiama `raise` per generare il segnale. Normalmente, i SIGINT sono generati al di fuori del programma quando qualcuno digita `<Ctrl>c` (Linux o Windows) o `<Comando>c` (macOS) per terminare l'esecuzione del programma. La gestione dei segnali può essere usata per intercettare un SIGINT ed evitare che il programma venga terminato.

✓ Autovalutazione

1. (*Completere*) Un evento, o _____, esterno asincrono può far sì che un programma termini prematuramente.

Risposta: segnale.

2. (*Scelta multipla*) Quale segnale standard è descritto come “un’operazione aritmetica erronea, come una divisione per zero o un’operazione che genera un overflow”?

- a) SIGILL
- b) SIGABRT
- c) SIGINT
- d) SIGFPE

Risposta: d.

15.8 Allocazione dinamica della memoria: funzioni `calloc` e `realloc`

Il Capitolo 12 ha introdotto la nozione di allocazione dinamica della memoria con l'uso della funzione `malloc`. Come abbiamo affermato nel Capitolo 12, gli array sono migliori delle liste collegate per quanto riguarda la velocità nell'ordinamento, nella ricerca e nell'accesso ai dati. Gli array sono normalmente **strutture di dati statiche** che non possono essere ridimensionate. La libreria di utilità generali (`<stdlib.h>`) fornisce le funzioni per l'allocazione dinamica della memoria `calloc` e `realloc` per creare **array dinamici** e modificarne le dimensioni.

Funzione `calloc`

La funzione `calloc` ("allocazione contigua")

```
void *calloc(size_t nmemb, size_t size);
```

alloca dinamicamente un array. I suoi due argomenti sono:

- il numero di elementi dell'array (`nmemb`);
- la dimensione di ogni elemento (`size`).

La funzione `calloc`, inoltre, inizializza gli elementi dell'array a zero. La funzione restituisce un puntatore alla memoria allocata o un puntatore `NULL` se la memoria non può essere allocata. La principale differenza tra `malloc` e `calloc` è che solo `calloc` azzerà la memoria che alloca.

Funzione `realloc`

La funzione `realloc`

```
void *realloc(void *ptr, size_t size);
```

cambia la dimensione di un oggetto allocato da una precedente chiamata a `malloc`, `calloc` o `realloc`. I contenuti originari dell'oggetto non sono modificati purché la quantità di memoria allocata sia più grande della quantità allocata in precedenza. Altrimenti, i contenuti rimangono non modificati fino alla dimensione del nuovo oggetto. I due argomenti della funzione sono:

- un puntatore all'oggetto originario (`ptr`);
- la nuova dimensione dell'oggetto (`size`).

Se `ptr` è `NULL`, `realloc` opera in maniera identica a `malloc`. Se `ptr` non è `NULL` e la dimensione è maggiore di zero, `realloc` cerca di allocare un nuovo blocco di memoria per l'oggetto. Se il nuovo spazio non può essere allocato, l'oggetto puntato da `ptr` rimane immutato. La funzione `realloc` restituisce un puntatore alla memoria riallocata o un puntatore `NULL` per indicare che la memoria non è stata riallocata.

✓ Autovalutazione

1. (*Completare*) Le funzioni per l'allocazione dinamica della memoria _____ e _____ della libreria di utilità generali (`<stdlib.h>`) creano e modificano array dinamici.

Risposta: `calloc` e `realloc`.

2. (*Scelta multipla*) Quale delle seguenti affermazioni è *falsa*?

- a) La funzione `calloc` alloca dinamicamente memoria per un array.
- b) I parametri `size_t nmemb` e `size_t size` della funzione `calloc` rappresentano il numero di elementi del nuovo array e la dimensione di ciascun elemento.
- c) La funzione `calloc` inizializza a zero gli elementi di un array allocato dinamicamente. La funzione restituisce un puntatore alla memoria allocata, o `NULL` se non è stato possibile allocare memoria.
- d) Le funzioni `malloc` e `calloc` azzerano la memoria che allocano.

Risposta: d) è *falsa*. In realtà, solo `calloc` azzerà la memoria che alloca.

3. (*Vero/Falso*) Se il primo argomento di realloc è NULL, realloc opera in maniera identica a malloc. Altrimenti, se l'argomento dimensione di realloc è maggiore di zero, realloc cerca di allocare un nuovo blocco di memoria. Se il nuovo spazio non può essere allocato, l'oggetto puntato dal primo argomento della funzione rimane immutato. La funzione restituisce un puntatore alla memoria riallocata o un puntatore NULL per indicare che la memoria non è stata riallocata.

Risposta: *Vero*.

15.9 goto: salto non condizionato

Abbiamo posto l'accento sull'importanza dell'uso delle tecniche di programmazione strutturata per costruire software affidabile, facile da correggere, mantenere e modificare. In alcuni casi, le prestazioni sono più importanti della stretta aderenza alle tecniche della programmazione strutturata. In questi casi è possibile usare alcune tecniche di programmazione non strutturata. Per esempio, possiamo usare break per terminare l'esecuzione di un'istruzione di iterazione prima che la condizione di continuazione del ciclo diventi falsa. Questo risparmia le iterazioni non necessarie del ciclo se il compito è completato *prima* del termine del ciclo.

Un altro esempio di programmazione non strutturata è l'**istruzione goto**, un salto non condizionato. L'istruzione goto altera il flusso di controllo, continuando l'esecuzione con la prima istruzione dopo l'**etichetta** specificata nell'istruzione. Un'etichetta è un identificatore seguito da due punti (:) e deve comparire nella *stessa* funzione dell'istruzione goto che si riferisce a essa. Le etichette non devono essere univoche tra le funzioni.

Dimostrazione di goto

Il programma della Figura 15.5 usa istruzioni goto per ripetere un ciclo dieci volte e stampare ogni volta il valore del contatore. La riga 6 inizializza count a 1. L'etichetta start: è ignorata perché le etichette non compiono alcuna azione. La riga 9 verifica se count è maggiore di 10. Se è così, la riga 10 trasferisce il controllo dal goto alla prima istruzione dopo l'etichetta end: (riga 19). Altrimenti, le righe 13-14 stampano e incrementano count, e il controllo viene trasferito dal goto (riga 16) alla prima istruzione dopo l'etichetta start: (riga 9).

```

1 // fig15_05.c
2 // Uso dell'istruzione goto
3 #include <stdio.h>
4
5 int main(void) {
6     int count = 1; // inizializza count
7
8     start: // etichetta
9     if (count > 10) {
10         goto end;
11     }
12
13     printf("%d ", count);
14     ++count;
15
16     goto start; // vai a start alla riga 9
17
18 end: // etichetta
19     putchar('\n');
20 }
```

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Figura 15.5 Uso dell'istruzione goto.

Nel Capitolo 3 abbiamo affermato che è possibile scrivere un programma in termini di istruzioni di sequenza, selezione e ripetizione. Quando si seguono le regole della programmazione strutturata è possibile creare strutture di controllo profondamente annidate all'interno di una funzione da cui è difficile uscire efficacemente.

 Alcuni programmatore usano le istruzioni goto in tali situazioni per una rapida uscita da una struttura di controllo profondamente annidata. Ciò elimina la necessità di verificare le diverse condizioni per uscire da una struttura di controllo. Vi sono altre situazioni in cui il goto è proprio raccomandato (consultate, per esempio, la raccomandazione MEM12-C del CERT, "Consider using a Goto-Chain when leaving a function on error when using and releasing resources"). L'istruzione goto non è strutturata e può portare a programmi molto più difficili da correggere, mantenere e modificare.

✓ Autovalutazione

1. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è *falsa*?

- a) Le tecniche di programmazione strutturata aiutano a costruire software affidabile, facile da correggere, mantenere e modificare.
- b) In alcuni casi, le prestazioni sono più importanti della stretta aderenza alle tecniche della programmazione strutturata. In questi casi è possibile usare alcune tecniche di programmazione non strutturata.
- c) Possiamo usare break per terminare l'esecuzione di un'istruzione di iterazione anticipatamente. Questo risparmia le iterazioni non necessarie del ciclo se il compito è completato prima del termine del ciclo.
- d) Tutte le affermazioni precedenti sono *vere*.

Risposta: d.

2. (*Scelta multipla*) Quale delle seguenti affermazioni a), b) o c) è *falsa*?

- a) Un esempio di programmazione non strutturata è l'istruzione goto, un salto non condizionato.
- b) L'istruzione goto altera il flusso di controllo, continuando l'esecuzione con la prima istruzione dopo l'etichetta specificata nell'istruzione. Un'etichetta è un identificatore seguito da due punti (:).
- c) Le etichette devono essere univoche tra tutte le funzioni di un'applicazione. Un'etichetta che è il target di una particolare istruzione goto può apparire in qualsiasi funzione di un'applicazione.
- d) Tutte le affermazioni precedenti sono *vere*.

Risposta: c) è *falsa*. In realtà, le etichette non devono essere univoche tra le funzioni. Inoltre, un'etichetta che è il target di un'istruzione goto in una funzione deve apparire in quella funzione.

3. (*Vero/Falso*) Quando si seguono le regole della programmazione strutturata è possibile creare strutture di controllo profondamente annidate all'interno di una funzione da cui è difficile uscire efficacemente. Alcuni programmatore usano le istruzioni goto in tali situazioni per una rapida uscita da una struttura di controllo profondamente annidata. Ciò elimina la necessità di verificare le diverse condizioni per uscire da una struttura di controllo.

Risposta: *Vero*.

15.10 Riepilogo

Paragrafo 15.2 Liste di argomenti di lunghezza variabile

- Il file di intestazione <stdarg.h> fornisce funzionalità per costruire funzioni con liste di argomenti di lunghezza variabile.
- Un'ellissi (...) nel prototipo di una funzione indica un numero variabile di argomenti.
- Il tipo va_list è adatto a contenere le informazioni necessarie alle macro va_start, va_arg e va_end.
- La macro va_start inizializza l'oggetto dichiarato con va_list per l'uso da parte delle macro va_arg e va_end.

- La macro `va_arg` viene espansa al valore e al tipo dell'argomento successivo della lista di argomenti di lunghezza variabile. Ogni invocazione a `va_arg` modifica l'oggetto dichiarato con `va_list` in modo che punti al successivo argomento.
- La macro `va_end` facilita un ritorno normale da una funzione alla cui lista di argomenti di lunghezza variabile faceva riferimento la macro `va_start`.

Paragrafo 15.3 Uso degli argomenti della riga di comando

- È possibile passare argomenti a `main` dalla riga di comando includendo i parametri `int argc` e `char *argv[]` nella lista dei parametri di `main`. Il parametro `argc` riceve il numero di argomenti della riga di comando. Il parametro `argv` è un array di stringhe in cui sono memorizzati gli argomenti della riga di comando.

Paragrafo 15.4 Compilazione di programmi con più file sorgente

- La definizione di una funzione deve essere interamente contenuta in un solo file.
- Lo **specificatore della classe di memoria extern** indica che una variabile è definita in seguito nello stesso file o in un file differente del programma.
- Le **variabili globali** devono essere dichiarate in ogni file in cui sono usate.
- Un **prototipo di funzione** può estendere il campo d'azione di una funzione oltre il file in cui essa è definita.
- L'applicazione dello **specificatore della classe di memoria static** a una funzione o a una variabile globale impedisce che essa venga usata al di fuori del file corrente. Questo è chiamato **collegamento interno**. Le funzioni e le variabili globali non precedute da `static` hanno un **collegamento esterno** ed è possibile accedervi in altri file contenenti dichiarazioni o prototipi di funzioni appropriati.
- Lo specificatore `static` è comunemente usato per le funzioni di utilità che sono chiamate solo dalle funzioni in un particolare file.
- Se una funzione non è necessaria al di fuori di un particolare file, applicate `static` a essa per far valere il principio del privilegio minimo.

Paragrafo 15.5 Terminazione di programmi con exit e atexit

- La funzione `exit` provoca la terminazione di un programma.
- La funzione `atexit` registra una funzione da chiamare quando il programma termina raggiungendo la fine di `main` o quando viene invocata `exit`.
- La funzione `atexit` riceve come argomento un puntatore a una funzione. Le funzioni chiamate al termine del programma non possono avere argomenti e non possono restituire un valore.
- La funzione `exit` riceve un argomento, normalmente la costante simbolica `EXIT_SUCCESS` o la costante simbolica `EXIT_FAILURE`.
- Quando viene chiamata la funzione `exit`, le funzioni registrate con `atexit` vengono invocate nell'ordine inverso rispetto a quello della loro registrazione.

Paragrafo 15.6 Suffissi per letterali interi e in virgola mobile

- I suffissi per valori interi e in virgola mobile consentono di specificare i tipi di costanti intere o in virgola mobile. I suffissi interi sono `u` o `U` per un intero `unsigned`, `l` o `L` per un intero `long`, e `ul` o `UL` per un intero `unsigned long`. Il tipo di una costante intera senza suffisso è determinato dal primo tipo in grado di memorizzare un valore di quella dimensione (`int`, poi `long int`, poi `unsigned long int` ecc.). I suffissi in virgola mobile sono `f` o `F` per un `float`, e `l` o `L` per un `long double`. Una costante in virgola mobile senza suffisso è di tipo `double`.

Paragrafo 15.7 Gestione dei segnali

- La libreria per la gestione dei segnali consente di intercettare eventi inattesi tramite la funzione `signal`.
- La funzione `signal` riceve due argomenti: un numero di segnale intero e un puntatore alla funzione per la gestione del segnale.
- È possibile generare segnali anche con la funzione `raise` e un argomento intero.

Paragrafo 15.8 Allocazione dinamica della memoria: funzioni `calloc` e `realloc`

- La libreria di utilità generali (`<stdlib.h>`) fornisce funzioni per l'allocazione dinamica della memoria: `calloc` e `realloc` per la creazione e il ridimensionamento di array dinamici.
- La funzione `calloc` alloca memoria per un array. Riceve il numero di elementi dell'array e la dimensione di ogni elemento, e inizializza a zero gli elementi dell'array. Restituisce un puntatore alla memoria allocata oppure un puntatore `NULL` se la memoria non è stata allocata.
- La funzione `realloc` cambia la dimensione di un oggetto allocato da una precedente chiamata a `malloc`, `calloc` o `realloc`. I contenuti dell'oggetto originario non sono modificati, a condizione che la quantità di memoria allocata sia più grande della quantità allocata in precedenza.
- La funzione `realloc` riceve un puntatore all'oggetto originario e la nuova dimensione dell'oggetto. Se `ptr` è `NULL`, `realloc` opera in maniera identica a `malloc`. Altrimenti, se `ptr` non è `NULL` e la dimensione è maggiore di zero, `realloc` cerca di allocare un nuovo blocco di memoria per l'oggetto. Se il nuovo spazio non può essere allocato, l'oggetto puntato da `ptr` rimane immutato. La funzione `realloc` restituisce un puntatore alla memoria riallocata o un puntatore a `NULL` per indicare che la memoria non è stata riallocata.

Paragrafo 15.9 goto: salto non condizionato

- L'istruzione `goto` altera il flusso di controllo di un programma. L'esecuzione del programma continua alla prima istruzione dopo l'etichetta specificata nell'istruzione `goto`.
- Un'etichetta è un identificatore seguito da due punti (:). Un'etichetta deve comparire nella stessa funzione dell'istruzione `goto` che si riferisce a essa.

Esercizio di autovalutazione

- 15.1 Riempite gli spazi vuoti in ognuna delle seguenti frasi.
- a) Una _____ nella lista dei parametri di una funzione indica che la funzione può ricevere un numero variabile di argomenti.
 - b) La macro _____ deve essere invocata prima che si possa accedere agli argomenti in una lista di argomenti di lunghezza variabile.
 - c) La macro _____ accede ai singoli argomenti di una lista di argomenti di lunghezza variabile.
 - d) La macro _____ facilita un normale ritorno da una funzione alla cui lista di argomenti di lunghezza variabile faceva riferimento la macro `va_start`.
 - e) L'argomento _____ di `main` riceve il numero di argomenti in una riga di comando.
 - f) L'argomento _____ di `main` memorizza gli argomenti di una riga di comando come stringhe di caratteri.
 - g) La funzione _____ forza un programma a terminare l'esecuzione.
 - h) La funzione _____ registra una funzione da chiamare alla normale terminazione di un programma.
 - i) Un _____ per letterali interi o in virgola mobile può essere aggiunto a una costante intera o in virgola mobile per specificare il tipo esatto della costante.
 - j) La funzione _____ può essere usata per intercettare eventi inattesi.
 - k) La funzione _____ genera un segnale dall'interno di un programma.
 - l) La funzione _____ alloca dinamicamente la memoria per un array e inizializza a zero gli elementi.
 - m) La funzione _____ modifica la dimensione di un blocco di memoria dinamica allocata in precedenza.

Risposte all'esercizio di autovalutazione

15.1 a) ellissi (...). b) va_start. c) va_arg. d) va_end. e) argc. f) argv. g) exit. h) atexit. i) suffisso. j) signal. k) raise. l) calloc. m) realloc.

Esercizi

15.2 (*Lista argomenti di lunghezza variabile: calcolo del prodotto*) Scrivete un programma che calcoli il prodotto di una serie di interi che vengono passati alla funzione product usando una lista di argomenti di lunghezza variabile. Provate la vostra funzione con diverse chiamate, ognuna con un numero differente di argomenti.

15.3 (*Stampare gli argomenti di una riga di comando*) Scrivete un programma che stampi gli argomenti della riga di comando del programma.

15.4 (*Ordinamento di interi*) Scrivete un programma che ordini un array di interi in modo crescente o decrescente. Usate gli argomenti della riga di comando per passare l'argomento -a per indicare l'ordine crescente o l'argomento -d per indicare l'ordine decrescente. [Nota: questo è il formato standard per passare le opzioni a un programma in UNIX.]

15.5 (*Gestione dei segnali*) Leggete la documentazione per il vostro compilatore per determinare quali segnali sono supportati dalla libreria per la gestione dei segnali (<signal.h>). Scrivete un programma che contenga i gestori di segnali per i segnali standard SIGABRT e SIGINT. Il programma deve intercettare questi segnali chiamando la funzione abort per generare un segnale di tipo SIGABRT e chiedendo all'utente di digitare <Ctrl>c o <Comando> C per generare un segnale di tipo SIGINT.

15.6 (*Allocazione dinamica di array*) Scrivete un programma che allochi in maniera dinamica un array di interi. La dimensione dell'array va inserita tramite tastiera. Agli elementi dell'array vanno assegnati valori inseriti anch'essi tramite tastiera. Stampate i valori dell'array. In seguito, riallocate la memoria per l'array con dimensione pari alla metà del numero corrente di elementi. Stampate i valori che rimangono nell'array per confermare che essi corrispondono alla prima metà dei valori dell'array originario.

15.7 (*Argomenti della riga di comando*) Scrivete un programma che riceva due nomi di file come argomenti della riga di comando, legga i caratteri dal primo file uno alla volta e li scriva in ordine inverso sul secondo file.

APPENDICE

A

Tabella di precedenza degli operatori

Gli operatori sono mostrati dall'alto in basso in ordine decrescente di precedenza.

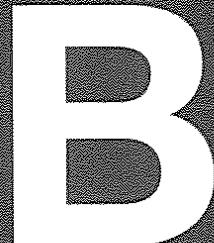
Operatore	Tipo	Associatività
()	parentesi (operatore di chiamata di funzione)	da sinistra a destra
[]	indice di array	
.	selezione di membro tramite oggetto	
->	selezione di membro tramite puntatore	
++	post-incremento unario	
--	post-decremento unario	
++	preincremento unario	da destra a sinistra
--	predecremento unario	
+	più unario	
-	meno unario	
!	negazione logica unaria	
-	complemento bit a bit unario	
(<i>tipo</i>)	cast unario in stile C	
*	dereferenziazione	
&	indirizzo	
sizeof	determina la dimensione in byte	
*	moltiplicazione	da sinistra a destra
/	divisione	
%	modulo	
+	addizione	da sinistra a destra
-	sottrazione	
<<	spostamento a sinistra bit a bit	da sinistra a destra
>>	spostamento a destra bit a bit	
<	relazionale minore di	da sinistra a destra
<=	relazionale minore o uguale a	
>	relazionale maggiore di	
>=	relazionale maggiore o uguale a	
==	relazionale uguale a	da sinistra a destra
!=	relazionale non uguale a	
&	AND bit a bit	da sinistra a destra
^	OR esclusivo bit a bit	da sinistra a destra
	OR inclusivo bit a bit	da sinistra a destra





&&	AND logico	da sinistra a destra
	OR logico	da sinistra a destra
? :	condizionale ternario	da destra a sinistra
=	assegnazione	da destra a sinistra
+=	assegnazione di addizione	
-=	assegnazione di sottrazione	
*=	assegnazione di moltiplicazione	
/=	assegnazione di divisione	
%=	assegnazione di modulo	
&=	assegnazione di AND bit a bit	
^=	assegnazione di OR esclusivo bit a bit	
=	assegnazione di OR inclusivo bit a bit	
<<=	assegnazione di spostamento a sinistra bit a bit	
>>=	assegnazione di spostamento a destra bit a bit	
,	virgola	da sinistra a destra

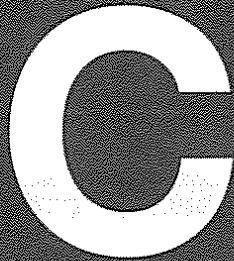
APPENDICE



Insieme di caratteri ASCII

Nella seguente tabella, le cifre nella colonna di sinistra sono le cifre a sinistra dell'equivalente decimale del codice carattere (0-127) e le cifre nella riga superiore sono le cifre a destra dell'equivalente decimale del codice carattere. Per esempio, il codice carattere per "A" nella riga numero 6 e nella colonna numero 5 è 65 e il codice carattere per "&" nella riga numero 3 e nella colonna numero 8 è 38.

Insieme dei caratteri ASCII												
	0	1	2	3	4	5	6	7	8	9		
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht		
1	lf	vt	ff	cr	so	s1	dle	dc1	dc2	dc3		
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs		
3	rs	us	sp	!	"	#	\$	%	&	'		
4	()	*	+	,	-	.	/	@	1		
5	2	3	4	5	6	7	8	9	:	:		
6	<	=	>	?	@	A	B	C	D	E		
7	F	G	H	I	J	K	L	M	N	O		
8	P	Q	R	S	T	U	V	W	X	Y		
9	Z	[\]	^	_	'	a	b	c		
10	d	e	f	g	h	i	j	k	l	m		
11	n	o	p	q	r	s	t	u	v	w		
12	x	y	z	{		}	-	del				



Multithreading/ multicore e altri argomenti di C18/C11/C99

Obiettivi

- Comprendere gli scopi del C18.
- Conoscere i file di intestazione aggiunti in C99 e C11/C18.
- Inizializzare array e strutture con inizializzatori designati.
- Usare il tipo di dati `bool` per creare variabili booleane i cui valori possono essere `true` o `false`.
- Eseguire operazioni aritmetiche su variabili complesse.
- Utilizzare le caratteristiche di multithreading per migliorare le prestazioni sugli attuali sistemi multicore.

C.1 Introduzione

Gli standard C99 (1999) e C11 (2011) ridefiniscono ed espandono le funzionalità del C. Dopo il C11, c'è stata solo un'altra nuova versione, il C18¹ (2018), che "ha risolto i problemi del C11 senza introdurre nuove funzionalità del linguaggio".² Alcune delle caratteristiche aggiunte dagli standard C99 e C11/C18 sono state definite come opzionali. Prima di usare le funzionalità qui illustrate, verificate che il vostro compilatore le supporti. Il nostro obiettivo è introdurre queste funzionalità e fornire risorse per ulteriori approfondimenti.

Spiegheremo con esempi di codice completo e con frammenti di codice gli inizializzatori designati, i letterali composti, il tipo `bool` e i numeri complessi. Forniremo brevi spiegazioni per ulteriori caratteristiche, inclusi i puntatori ristretti, la divisione intera affidabile, i membri array flessibili, il tipo `math` generico e le funzioni `inline`.

Esamineremo le funzionalità di C11/C18, incluso il supporto Unicode® migliorato, lo specificatore di funzione `_Noreturn`, le espressioni di tipo generico, la funzione `quick_exit`, il controllo dell'allineamento in memoria, le asserzioni statiche, l'analizzabilità e i tipi in virgola mobile.

Multithreading in C11/C18

Una caratteristica chiave di questa appendice è l'introduzione al multithreading del Paragrafo C.9. Negli attuali sistemi *multicore*, l'hardware può impiegare più processori (*core*) per lavorare su differenti parti di un'attività. Questo consente un'esecuzione più veloce delle attività (e del programma). Per trarre vantaggio dall'architettura multicore da programmi in C è necessario scrivere applicazioni multithreaded. Quando un programma divide le attività in thread separati, un sistema multicore può eseguire quei thread in parallelo (ovvero simultaneamente). Il Paragrafo C.9 prima mostra due calcoli intensivi eseguiti in sequenza. Poi questi calcoli vengono separati in due thread per dimostrare il significativo miglioramento delle prestazioni dovuto all'esecuzione in parallelo dei thread su più core.

1. ISO/IEC 9899:2018, Information technology — Programming languages — C, <https://www.iso.org/standard/74528.html>.

2. [https://en.wikipedia.org/wiki/C18_\(C_standard_revision\)](https://en.wikipedia.org/wiki/C18_(C_standard_revision)). Anche http://www.iso-9899.info/wiki/The_Standard.

C.2 File di intestazione aggiunti nel C99

Nella tabella seguente sono elencati i file di intestazione della Libreria Standard aggiunti in C99, che rimangono disponibili in C11/C18. Discuteremo i nuovi file di intestazione di C11/C18 nel Paragrafo C.8.1.

File di intestazione	Spiegazione
<complex.h>	Contiene supporto per numeri complessi (vedi Paragrafo C.5).
<fenv.h>	Fornisce informazioni sull'ambiente e sulle funzionalità dell'implementazione in C dei numeri in virgola mobile.
<inttypes.h>	Definisce tipi interi portabili e fornisce gli specificatori di formato per essi.
<stdbool.h>	Contiene le macro che definiscono <code>bool</code> , <code>true</code> e <code>false</code> , utilizzati per le variabili booleane (vedi Paragrafo C.4).
<stdint.h>	Definisce tipi interi estesi e relative macro.
<tgmath.h>	Fornisce le macro per i tipi generici che permettono alle funzioni di <math.h> di essere utilizzate con una varietà di tipi di parametri (vedi Paragrafo C.7).

C.3 Inizializzatori designati e letterali composti

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 10.3.]

Gli inizializzatori designati permettono di inizializzare gli elementi di un array tramite indice e i membri di `union` o `struct` tramite nome. Il programma della Figura C.1 mostra che possiamo usare inizializzatori designati per inizializzare specifici elementi di un array.

```

1 // figC_01.c
2 // Inizializzare specifici elementi di un array con inizializzatori designati...
3 #include <stdio.h>
4
5 int main(void) {
6     int values[5] = {
7         [0] = 123, // inizializza l'elemento 0
8         [4] = 456 // inizializza l'elemento 4
9     }; // e' richiesto il punto e virgola
10
11    // stampa il contenuto dell'array
12    printf("values: ");
13
14    for (size_t i = 0; i < 5; ++i) {
15        printf("%d ", values[i]);
16    }
17
18    puts("");
19 }
```

```
values: 123 0 0 0 456
```

Figura C.1 Inizializzare specifici elementi di un array con inizializzatori designati.

Le righe 6-9 definiscono un array e inizializzano i suoi elementi 0 e 4 all'interno di parentesi graffe. Osservate la sintassi. Ogni inizializzatore è separato dal successivo da una virgola. La parentesi graffa di chiusura della lista degli inizializzatori deve essere seguita da un punto e virgola. Gli elementi che non vengono inizializzati esplicitamente sono inizializzati implicitamente a zero.

Letterali composti

Potete usare una lista di inizializzatori per creare un array anonimo, struct o union. Questo costrutto è noto come **letterale composto**. Per esempio, potete passare l'array della Figura C.1 a una funzione senza doverlo preventivamente dichiarare, come in

```
demoFunction((int [5]) {[0] = 1, [4] = 2});
```

Il programma della Figura C.2 usa i letterali composti come inizializzatori designati per elementi specifici in un array di struct. Le righe 12 e 13 usano ciascuna un inizializzatore designato per inizializzare esplicitamente un elemento struct nell'array. Per esempio, alla riga 12, l'espressione seguente è un letterale composto che crea un oggetto struct anonimo di tipo struct twoInt:

```
{.x = 1, .y = 2}
```

I membri x e y di quell'oggetto sono inizializzati a 1 e 2. Gli inizializzatori designati per i membri struct e union scrivono il nome di ogni membro preceduto da un punto (.).

```

1 // figC_02.c
2 // Inizializzare membri di struct con inizializzatori designati.
3 #include <stdio.h>
4
5 struct twoInt { // dichiara una struttura di due interi
6     int x;
7     int y;
8 };
9
10 int main(void) {
11     struct twoInt a[5] = {
12         [0] = {.x = 1, .y = 2},
13         [4] = {.x = 10, .y = 20}
14     };
15
16     // stampa il contenuto dell'array
17     printf("%2s%5s\n", "x", "y");
18
19     for (size_t i = 0; i < 5; ++i) {
20         printf("%2d%5d\n", a[i].x, a[i].y);
21     }
22 }
```

x	y
1	2
0	0
0	0
0	0
10	20

Figura C.2 Inizializzare membri di struct con inizializzatori designati.

Le righe 11-14 sono più immediate rispetto al seguente codice eseguibile, che non usa inizializzatori designati:

```
struct twoInt a[5];

a[0].x = 1;
a[0].y = 2;
a[4].x = 10;
a[4].y = 20;
```

 Utilizzando inizializzatori anziché assegnazioni in fase di esecuzione è possibile migliorare i tempi di avvio del programma.

C.4 Il tipo bool

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 3.6.]

Il tipo booleano, `Bool`, può assumere soltanto un valore 0 o un valore 1. Ricordate che nelle condizioni del C, zero rappresenta *falso*, e ogni valore diverso da zero rappresenta *vero*. Assegnare *qualsiasi* valore diverso da zero a un `_Bool` lo imposta a 1. Il file di intestazione `<stdbool.h>` definisce le macro `bool`, `false` e `true`. Queste macro sostituiscono `bool` con la parola chiave `_Bool`, `false` con 0, e `true` con 1. Il programma della Figura C.3 usa una funzione denominata `isEven` (righe 28-35) che restituisce il valore `true` di tipo `bool` se l'argomento della funzione è pari, e `false` se è dispari.

```
1 // figC_03.c
2 // Utilizzare bool, true e false.
3 #include <stdio.h>
4 #include <stdbool.h> // consente l'uso di bool, true e false
5
6 bool isEven(int number); // prototipo di funzione
7
8 int main(void) {
9     // ciclo per 2 input
10    for (int i = 0; i < 2; ++i) {
11        printf("Enter an integer: ");
12        int input = 0; // valore inserito dall'utente
13        scanf("%d", &input);
14
15        bool valueIsEven = isEven(input); // determina se il valore e' pari
16
17        // determina se il valore e' pari
18        if (valueIsEven) {
19            printf("%d is even\n\n", input);
20        }
21        else {
22            printf("%d is odd\n\n", input);
23        }
24    }
25 }
26
27 // isEven restituisce true se number e' pari
28 bool isEven(int number) {
29     if (number % 2 == 0) { // number e' divisibile per 2?
30         return true;
31     }
32     else {
```

```

33     return false;
34 }
35 }
```

```
Enter an integer: 34
34 is even
```

```
Enter an integer: 23
23 is odd
```

Figura C.3 Utilizzare `bool`, `true` e `false`.

La riga 15 dichiara la variabile `valueIsEven` di tipo `bool` e passa l'input dell'utente alla funzione `isEven`, che restituisce un valore di tipo `bool`. La riga 29 determina se l'argomento è divisibile per 2. Se lo è, la riga 30 restituisce `true`; altrimenti, la riga 33 restituisce `false`. Il risultato è assegnato alla variabile `valueIsEven` di tipo `bool` nella riga 15. Se `valueIsEven` è `true`, la riga 19 stampa una stringa che indica che il valore è pari. Se `valueIsEven` è `false`, la riga 22 stampa una stringa che indica che il valore è dispari. Il corpo della funzione `isEven` può essere scritto più sinteticamente come

```
return number % 2 == 0;
```

ma abbiamo voluto mostrare le macro `true` e `false` del file di intestazione `<stdbool.h>`.

C.5 Numeri complessi

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 5.3.]

Il C99 ha introdotto le funzionalità di supporto per i numeri complessi e l'aritmetica complessa. Il programma della Figura C.4 mostra operazioni di base con numeri complessi. Abbiamo compilato ed eseguito questo programma usando Xcode di Apple. Visual C++ di Microsoft supporta solo funzionalità per numeri complessi definite dallo standard C++, non quelle del C.

```

1 // figC_04.c
2 // Operazioni con numeri complessi.
3 #include <complex.h> // per i tipi complessi e le funzioni matematiche
4 #include <stdio.h>
5
6 int main(void) {
7     double complex a = 3.0 + 2.0 * I;
8     double complex b = 2.7 + 4.9 * I;
9
10    printf("a is %.1f + %.1fi\n", creal(a), cimag(a));
11    printf("b is %.1f + %.1fi\n", creal(b), cimag(b));
12
13    double complex sum = a + b; // calcola l'addizione complessa
14    printf("a + b is: %.1f + %.1fi\n", creal(sum), cimag(sum));
15
16    double complex difference = a - b; // calcola la sottrazione complessa
17    printf("a - b is: %.1f + %.1fi\n", creal(difference), cimag(difference));
18
19    double complex product = a * b; // calcola la moltiplicazione complessa
20    printf("a * b is: %.1f + %.1fi\n", creal(product), cimag(product));
21
22    double complex quotient = a / b; // calcola la divisione complessa
```

```

23     printf("a / b is: %.1f + %.1fi\n", creal(quotient), cimag(quotient));
24
25     double complex power = cpow(a, 2.0); // calcola l'esponenziazione complessa
26     printf("a ^ b is: %.1f + %.1fi\n", creal(power), cimag(power));
27 }

a is 3.0 + 2.0i
b is 2.7 + 4.9i
a + b is: 5.7 + 6.9i
a - b is: 0.3 + -2.9i
a * b is: -1.7 + 20.1i
a / b is: 0.6 + -0.3i
a ^ b is: 5.0 + 12.0i

```

Figura C.4 Operazioni con numeri complessi.

Per usare i numeri **complessi**, includete il file di intestazione `<complex.h>` (riga 3). Questo espanderà la macro `complex` producendo la parola chiave `_Complex`: un tipo che riserva un array di esattamente due elementi, corrispondenti alla *parte reale* e alla *parte immaginaria* di un numero complesso. Definite le variabili `complex` come mostrato nelle righe 7, 8, 13, 16, 19, 22 e 25. Definiamo ognuna delle variabili come tipo `double complex`, per indicare che le parti reale e immaginaria del numero complesso sono memorizzate come valori di tipo `double`. Il linguaggio C supporta anche i tipi `float complex` o `long double complex`.

Gli operatori aritmetici si applicano anche ai numeri complessi e il file di intestazione `<complex.h>` fornisce funzioni matematiche aggiuntive, come `cpow` alla riga 25. Potete anche usare gli operatori `!`, `++`, `--`, `&&`, `||`, `==`, `!=` e `&` unario con numeri complessi.

Le righe 13-26 stampano i risultati di varie operazioni aritmetiche. Alla *parte reale* e alla *parte immaginaria* di un numero complesso si può accedere, rispettivamente, con le funzioni `creal` e `cimag`, come mostrato alla riga 10. Nella stringa di output della riga 26 usiamo il simbolo `^` per indicare l'esponenziazione.

C.6 Macro con liste di argomenti di lunghezza variabile

Le macro possono avere liste di argomenti di lunghezza variabile. Questa caratteristica consente l'implementazione di macro "involturco" attorno a funzioni come `printf`. Per esempio, per aggiungere automaticamente il nome del file corrente a un'istruzione di debug, è possibile definire una macro come segue:

```
#define DEBUG(...) printf(__FILE__ ":" __VA_ARGS__)
```

Questa macro `DEBUG` riceve un numero variabile di argomenti, come indicato dal simbolo `...` nella lista degli argomenti. Come le funzioni, il simbolo `...` deve essere l'ultimo argomento. Diversamente dalle funzioni, il simbolo `...` può essere l'*unico* argomento della macro. L'identificatore `__VA_ARGS__`, che inizia e termina con due trattini bassi, è un segnaposto per la lista di argomenti di lunghezza variabile. Supponendo che questa macro compaia nel file `file.c`, il preprocessore sostituisce la seguente invocazione di macro:

```
DEBUG("x = %d, y = %d\n", x, y);
```

con

```
printf("file.c ":" " "x = %d, y = %d\n", x, y);
```

Ricordate che le stringhe separate da caratteri di spaziatura sono concatenate durante la preelaborazione, quindi le tre stringhe letterali verranno combinate assieme per formare il primo argomento della funzione `printf`.

C.7 Altre caratteristiche del C99

Nel seguito presentiamo brevi panoramiche di ulteriori caratteristiche del C99. Queste includono parole chiave, funzionalità del linguaggio e integrazioni della Libreria Standard.

C.7.1 Limiti minimi delle risorse per i compilatori

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 15.4.]

Prima del C99, lo standard richiedeva che le implementazioni del C supportassero identificatori di

- non meno di 31 caratteri per identificatori con collegamento interno;
- non meno di 6 caratteri per identificatori con collegamento esterno (Paragrafo 15.4).

Il C99 aumenta questi limiti a 63 caratteri per identificatori con collegamento interno e a 31 caratteri per identificatori con collegamento esterno. Questi sono solo limiti inferiori. I compilatori sono liberi di supportare identificatori con più caratteri di questi limiti. Per maggiori informazioni, consultate il Paragrafo 5.2.4.1 dello standard C18.

Il C pone dei limiti minimi anche su molte caratteristiche del linguaggio. Per esempio, ai compilatori viene richiesto di supportare almeno 1023 membri in un costrutto `struct`, `enum` o `union`, e almeno 127 parametri per una funzione. Per maggiori informazioni su altri limiti, consultate il Paragrafo 5.2.4.1 dello standard C18.

C.7.2 La parola chiave `restrict`

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 7.5.]

La parola chiave `restrict` è usata per dichiarare un puntatore ristretto che dovrebbe avere accesso esclusivo a una regione della memoria. Agli oggetti accessibili attraverso un puntatore ristretto non si può avere accesso per mezzo di altri puntatori, eccetto quando il valore di quei puntatori è derivato dal valore dello stesso puntatore ristretto, per esempio assegnando un puntatore qualificato `restrict` a un puntatore qualificato `non-restrict`.

Possiamo dichiarare un puntatore ristretto a un `int` come:

```
int *restrict ptr;
```

I puntatori ristretti consentono al compilatore di ottimizzare le modalità con cui il programma ha accesso alla memoria. Per esempio, la funzione `memcpy` della Libreria Standard è definita come segue:

```
void *memcpy(void *restrict s1, const void *restrict s2, size_t n);
```

La specifica della funzione `memcpy` stabilisce che questa non può essere usata per effettuare copie tra regioni che si sovrappongono in memoria. L'uso di puntatori ristretti permette al compilatore di ottimizzare l'operazione di copia copiando più byte in una volta, il che è più efficiente. Dichiарare un puntatore in maniera non corretta come ristretto quando un altro puntatore punta alla stessa regione della memoria può produrre un *comportamento indefinito*. Per maggiori informazioni, consultate il Paragrafo 6.7.3.1 dello standard C99.

C.7.3 Divisione intera affidabile

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 2.5.]

Nei primi compilatori C, il comportamento della divisione intera variava fra un'implementazione e l'altra. Alcune implementazioni arrotondavano un quoziente negativo nella direzione dell'infinito negativo, mentre altre lo arrotondavano nella direzione dello zero, ottenendo risposte diverse. Considerate la divisione di -28 per 5 . La risposta esatta è $-5,6$. Se arrotondiamo il quoziente nella direzione dello zero, otteniamo -5 . Se arrotondiamo $-5,6$ nella direzione dell'infinito negativo, otteniamo -6 . I compilatori C odierni scartano semplicemente la parte frazionaria (l'equivalente di arrotondare il quoziente nella direzione dello zero), rendendo i risultati della divisione di interi affidabili tra i sistemi. Per maggiori informazioni, consultate il Paragrafo 6.5.5 dello standard C.

C.7.4 Membri array flessibili

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 10.3.]

L'ultimo membro di un costrutto `struct` può essere un array di lunghezza non specificata, come in:

```
struct s {
    int arraySize;
    int array[];
};
```

Questo è chiamato **membro array flessibile** e viene dichiarato specificando le parentesi quadre vuote (`[]`) dopo il nome dell'array. Per allocare un costrutto `struct` con un membro array flessibile usate un codice del tipo:

```
int desiredSize = 5;
struct s *ptr;
ptr = malloc(sizeof(struct s) + sizeof(int) * desiredSize);
```

L'operatore `sizeof` ignora i membri array flessibili, così `sizeof(struct s)` restituisce la dimensione di tutti i membri in `struct` *eccetto* il membro array flessibile. Lo spazio extra che allochiamo con `sizeof(int) * desiredSize` è la dimensione dell'array flessibile.

Restrizioni per i membri array flessibili

Vi sono molte restrizioni sull'uso dei membri array flessibili.

- Un membro array flessibile può essere dichiarato solo come ultimo membro di un costrutto `struct`, quindi ogni costrutto `struct` può contenere al massimo un solo membro array flessibile.
- un array flessibile non può essere l'unico membro di un costrutto `struct`: il costrutto `struct` deve avere anche uno o più membri fissi.
- Un costrutto `struct` contenente un array flessibile potrebbe non essere un membro di un altro costrutto `struct`.
- Un costrutto `struct` con un membro array flessibile deve essere allocato dinamicamente. Non è possibile definire la dimensione del membro array flessibile al momento della compilazione.

Per maggiori informazioni, consultate il Paragrafo 6.7.2.1 dello standard C99.

C.7.5 Tipo math generico

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 5.3.]

Il file di intestazione `<tgmath.h>` del C99 fornisce macro di tipo generico per molte funzioni matematiche in `<math.h>`. Per esempio, dopo aver incluso `<tgmath.h>`, l'espressione `sin(x)` chiamerà:

- `sinf` (la versione `float` di `sin`) se `x` è un `float`;
- `sin` (che riceve un argomento `double`) se `x` è un `double`;
- `sinl` (la versione `long double` di `sin`) se `x` è un `long double`;
- una funzione `csin`, `csinfo` o `csinl` (le funzioni `sin` per tipi `complex`) se `x` è un tipo `complex`.

C11/C18 aggiunge ulteriori funzionalità generiche cui accenneremo nel seguito di questa appendice.

C.7.6 Funzioni inline

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 5.5.]

È possibile dichiarare funzioni inline mettendo la parola chiave `inline` prima della dichiarazione di una funzione, come in:

```
inline void someFunction();
```

- Questo può migliorare le prestazioni. Le chiamate di funzione richiedono tempo. Quando dichiariamo una funzione come `inline`, il programma potrebbe non chiamare più quella funzione. Invece, il compilatore ha la possibilità di sostituire ogni chiamata a una funzione `inline` con una copia del corpo del codice di quella funzione. Ciò migliora le prestazioni al momento dell'esecuzione, ma può aumentare la dimensione del programma. Dichiarare le funzioni come `inline` solo se sono brevi e chiamate frequentemente. Se modificate la definizione di una funzione `inline`, dovete ricompilare qualsiasi codice che chiama quella funzione. La dichiarazione `inline` è solo un suggerimento al compilatore, che può decidere di ignorarlo. I compilatori possono anche ottimizzare le prestazioni incorporando funzioni non dichiarate `inline`. Per maggiori informazioni, consultate il Paragrafo 6.7.4 dello standard C99.

C.7.7 Identificatore predefinito `_func_`

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 14.9.]

L'identificatore predefinito `_func_` è simile alle macro del preprocessore `_FILE_` e `_LINE_`. Quando usato nel corpo di una funzione, `_func_` è una stringa contenente il nome della funzione corrente. A differenza di `_FILE_` e `_LINE_`, `_func_` è una variabile reale, non una stringa letterale visibile in fase di preelaborazione. Quindi, non è possibile concatenare `_func_` con altri letterali durante la preelaborazione.

C.7.8 Macro `va_copy`

[Questo paragrafo può essere trattato in un corso dopo il Paragrafo 15.2.]

Il Paragrafo 15.2 ha introdotto il file di intestazione `<stdarg.h>` e le funzioni con liste di argomenti di lunghezza variabile. La macro `va_copy` riceve due `va_list` e copia il suo secondo argomento nel suo primo argomento. Ciò consente scorrimenti multipli di una lista di argomenti di lunghezza variabile senza ricominciare ogni volta dall'inizio.

C.8 Caratteristiche di C11/C18

C11/C18 affina ed espande le funzionalità del C. Alcune caratteristiche di C11/C18 sono considerate opzionali. Il Visual C++ di Microsoft fornisce solo un supporto parziale per le caratteristiche che sono state aggiunte in C99 e C11/C18.

C.8.1 File di intestazione di C11/C18

Nella seguente tabella sono elencati i file di intestazione della Libreria Standard che sono stati aggiunti nel C11.

File di intestazione	Spiegazione
<code><stdalign.h></code>	Fornisce i controlli per l'allineamento dei tipi.
<code><stdatomic.h></code>	Fornisce l'accesso senza possibili interruzioni agli oggetti utilizzati nel multithreading.
<code><stdnoreturn.h></code>	Funzioni senza ritorno.
<code><threads.h></code>	Libreria per i thread (vedi Paragrafo C.9).
<code><uchar.h></code>	Utilità per i caratteri UTF-16 e UTF-32.

C.8.2 Funzione `quick_exit`

Oltre a `exit` (Paragrafo 15.5) e `abort`, C11/C18 fornisce la funzione `quick_exit` (file di intestazione `<stdlib.h>`) per terminare un programma. Come `exit`, `quick_exit` riceve come argomento un valore che rappresenta uno stato di uscita, tipicamente `EXIT_SUCCESS` o `EXIT_FAILURE`, ma sono possibili anche altri

valori specifici per la particolare piattaforma. Il programma restituisce il valore dello stato di uscita all'ambiente chiamante per indicare se il programma ha terminato con successo o si è verificato un errore.

Quando viene chiamata, `quick_exit` può, a sua volta, chiamare fino a 32 altre funzioni per eseguire compiti di ripulitura. Registrerete queste funzioni, che devono restituire `void` e avere una lista di parametri `void`, con la funzione `at_quick_exit` (simile ad `atexit` del Paragrafo 15.5). Esse vengono chiamate in ordine inverso rispetto a quello col quale sono state registrate. La motivazione per le funzioni `quick_exit` e `at_quick_exit` è spiegata all'indirizzo

<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1327.htm>

C.8.3 Supporto Unicode®

Il processo di internazionalizzazione e localizzazione consiste nella creazione di software in grado di supportare diversi linguaggi parlati e requisiti specifici locali, come per esempio la stampa di formati monetari. L'insieme di caratteri **Unicode®** contiene caratteri per molti linguaggi e simboli del mondo.

C11/C18 include il supporto per entrambi gli insiemi di caratteri Unicode a *16 bit (UTF-16)* e *32 bit (UTF-32)*, che vi consente di internazionalizzare e localizzare le vostre app più facilmente. Il Paragrafo 6.4.5 dello standard C18 analizza le stringhe letterali Unicode. Il Paragrafo 7.28 dello standard illustra le caratteristiche del file di intestazione (`<uchar.h>`) per le nuove utilità Unicode, che include i nuovi tipi `char16_t` e `char32_t` rispettivamente per i caratteri UTF-16 e UTF-32.

C.8.4 Specificatore di funzione _Noreturn

Lo specificatore di funzione `_Noreturn` indica che una funzione non ritornerà alla sua funzione chiamante. Per esempio, la funzione `exit` (Paragrafo 15.5) termina un programma, per cui non ritorna alla sua funzione chiamante. Tali funzioni nella Libreria Standard del C sono adesso dichiarate con `_Noreturn`. Per esempio, gli standard C11/C18 presentano il prototipo della funzione `exit` come:

```
_Noreturn void exit(int status);
```

Se il compilatore sa che una funzione non ritorna, può eseguire varie ottimizzazioni ed emettere messaggi di errore se a una funzione `_Noreturn` è inavvertitamente richiesto di ritornare.

C.8.5 Espressioni di tipo generico

La parola chiave `_Generic` di C11/C18 fornisce un meccanismo utilizzabile per creare una macro (Capitolo 14) che può invocare diverse versioni specifiche per tipo di funzioni in base al tipo di argomento della macro. In C11/C18, `_Generic` si utilizza per implementare le funzionalità del file di intestazione (`<tgmath.h>`) per funzioni matematiche di tipo generico. Molte funzioni matematiche si presentano in versioni separate che ricevono argomenti `float`, `double` o `long double`. Per tali casi, vi è una macro che invoca automaticamente la corrispondente versione per il tipo specifico. Per esempio, la macro `ceil` invoca la funzione `ceilf` quando l'argomento è un `float`, `ceil` quando l'argomento è un `double` e `ceill` quando l'argomento è un `long double`. Il Paragrafo 6.5.1.1 dello standard C18 esamina i dettagli dell'uso di `_Generic`.

C.8.6 Annex L: analizzabilità e comportamento indefinito

I documenti degli standard C11/C18 definiscono le caratteristiche del linguaggio che i progettisti di compilatori devono implementare. A causa della straordinaria varietà delle piattaforme hardware e software e di altri elementi, vi è un certo numero di situazioni in cui lo standard specifica che il risultato di un'operazione è un comportamento indefinito. Ciò può suscitare problematiche di sicurezza e di affidabilità: ogni volta che vi è un comportamento indefinito accade qualcosa che potrebbe lasciare un sistema esposto ad attacchi o a malfunzionamenti. Il termine “comportamento indefinito” compare approssimativamente 50 volte nel documento dello standard C18.

I responsabili dell'Annex L opzionale di C11/C18 sono membri della CERT Division del Software Engineering Institute della Carnegie Mellon University:

<https://www.sei.cmu.edu/about/divisions/cert/index.cfm>

Essi hanno analizzato tutti i comportamenti indefiniti menzionati nel C standard e hanno scoperto che questi ricadono in due categorie:

- quelli per i quali gli implementatori di compilatori devono essere in grado di fare qualcosa di ragionevole per evitare serie conseguenze (noti come *comportamenti indefiniti limitati*);
- quelli per i quali gli implementatori non potrebbero fare alcunché di ragionevole (noti come *comportamenti indefiniti critici*).

Ne è risultato che la maggior parte dei comportamenti indefiniti appartiene alla prima categoria. David Keaton (un ricercatore del CERT Secure Coding Program) illustra le categorie nel seguente articolo:

https://insights.sei.cmu.edu/sei_blog/2012/06/improving-security-in-the-latest-c-programming-language-standard.html

L'Annex L degli standard C11/C18 identifica i comportamenti indefiniti critici. L'inclusione di questo annex come parte dello standard è un'opportunità per le implementazioni di compilatori. Un compilatore che è conforme all'Annex L può essere affidabile per fare qualcosa di ragionevole rispetto alla maggior parte dei comportamenti indefiniti che potrebbero essere stati ignorati in implementazioni precedenti. L'Annex L non garantisce ancora un comportamento ragionevole per comportamenti indefiniti critici. Un programma può determinare se l'implementazione è compatibile con l'Annex L tramite l'uso di direttive di compilazione condizionale (Paragrafo 14.5) per testare se è definita la macro `_STDC_ANALYZABLE_`.

C.8.7 Controllo dell'allineamento in memoria

Nel Capitolo 10 abbiamo discusso del fatto che le varie piattaforme di sistemi di elaborazione hanno differenti requisiti di allineamento ai confini di parole di memoria, il che può portare a oggetti `struct` che richiedono più memoria della somma delle dimensioni dei loro membri. C11/C18 consente di specificare i requisiti di allineamento ai confini per ogni tipo utilizzando le funzionalità del file di intestazione `<stdalign.h>`. `_Alignas` è usato per specificare i requisiti di allineamento. L'operatore `alignof` restituisce il requisito di allineamento per il suo argomento. La funzione `aligned_alloc` permette di allocare la memoria dinamicamente per un oggetto e di specificarne i requisiti di allineamento. Per maggiori dettagli, consultate il Paragrafo 6.2.8 del documento dello standard C18.

C.8.8 Asserzioni statiche

Nel Paragrafo 14.10 avete appreso che la macro `assert` del C testa il valore di un'espressione al momento dell'esecuzione. Se il valore della condizione è falso, `assert` stampa un messaggio di errore e chiama la funzione `abort` per terminare il programma. Questo è utile per fini di debugging. C11/C18 fornisce `_Static_assert` per le asserzioni riferite alla fase di compilazione, che testano espressioni costanti dopo l'esecuzione del preprocessore e al punto durante la compilazione in cui i tipi delle espressioni sono noti. Per maggiori dettagli, consultate il Paragrafo 6.7.10 del documento dello standard C18.

C.8.9 Tipi in virgola mobile

I compilatori che supportano C11/C18 offrono un supporto opzionale per lo standard aritmetico in virgola mobile IEC 60559. Tra le sue caratteristiche, IEC 60559 definisce come deve essere eseguito il calcolo aritmetico in virgola mobile per essere certi di ottenere sempre lo stesso risultato in tutte le implementazioni, sia che i calcoli vengano eseguiti dall'hardware, dal software o da entrambi. Per maggiori informazioni su questo standard consultate:

http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57469

C.9 Caso pratico: prestazioni con sistemi multithreading e multicore

Sarebbe bello se potessimo concentrare la nostra attenzione su una sola attività alla volta ed eseguirla in maniera ottimale. Ma questo è difficile da realizzare in un mondo complesso nel quale tantissime cose accadono contemporaneamente. Questo paragrafo presenta le funzionalità del C utili per creare e gestire più attività. Come mostreremo, ciò può migliorare significativamente le prestazioni e i tempi di risposta di un programma.

Concorrenza vs parallelismo

Quando diciamo che due attività vengono svolte **concorrentemente**, si intende che entrambe *progrediscono* allo stesso tempo. Sino ai primi anni 2000, la maggior parte dei computer aveva un solo processore. I sistemi operativi di questo tipo di computer eseguono contemporaneamente varie attività passando rapidamente da una all'altra, effettuando una piccola parte di ciascuna prima di passare alla successiva, in modo che tutte le attività continuino a progredire. Per esempio, comunemente un computer esegue molte attività in contemporanea, quali compilare un programma, inviare un file alla stampante, ricevere e-mail, postare un tweet, caricare un video su YouTube, caricare una foto su Facebook o Instagram, ecc.

 Quando diciamo che due attività vengono eseguite **in parallelo**, si intende che esse *vengono eseguite davvero simultaneamente*. In questo senso, si può considerare il parallelismo come una sottocategoria della concorrenza. Il corpo umano esegue una moltitudine di operazioni in parallelo. Per esempio, la respirazione, la circolazione del sangue, la digestione, il pensare e il camminare possono avvenire in parallelo; allo stesso modo possono agire in parallelo anche tutti i sensi (vista, udito, tatto, olfatto e gusto).

Non si conosce con esattezza la potenza del cervello umano, ma diversi articoli affermano che equivalga a quella di 100 miliardi di "processori"^{3,4,5} e abbiamo trovato un articolo secondo il quale il cervello possiede l'equivalente di "cinque milioni di processori moderni con chip da 2000 milioni di transistor".⁶ I computer multicore attuali hanno processori multipli che possono eseguire attività in parallelo.

Concorrenza nel C

I programmi in linguaggio C possono avere più **thread di esecuzione**, ognuno con la propria pila delle chiamate delle funzioni e il proprio contatore del programma (che tiene traccia dell'istruzione successiva da eseguire), consentendo a quel thread di essere eseguito contemporaneamente ad altri thread. Questa funzionalità è denominata **multithreading**.

 Un problema delle applicazioni a thread singolo è che le attività di lunga durata devono essere completate prima che le altre possano iniziare, e questo può portare a una scarsa reattività. Nelle applicazioni multithreaded, i thread possono essere distribuiti su diversi core disponibili in modo che più attività vengano eseguite in parallelo, consentendo all'applicazione di operare con più efficienza. Il multithreading può anche migliorare le prestazioni su sistemi a processore singolo: quando un thread non può continuare (perché, per esempio, è in attesa che si verifichi un evento, come la scadenza di un timer o il completamento di un'operazione di I/O), un altro thread può usare il processore.

In un sistema a processore singolo con multithreading si possono eseguire più thread contemporaneamente, ma non in parallelo. In un sistema multicore, con il multithreading si possono eseguire alcuni thread contemporaneamente ed eseguirne altri realmente in parallelo.

3. "How Many Supercomputers Would Fit Inside Your Brain?". Accesso 4 dicembre 2020. <https://fountainmagazine.com/2016/issue-111-may-june-2016/how-many-supercomputers-would-fit-inside-your-brain>.

4. "When compared to a computer CPU, is human brain single-core or multi-core?". Accesso 4 dicembre 2020. <https://www.quora.com/When-compared-to-a-computer-CPU-is-human-brain-single-core-or-multi-core/answer/Frank-Heile>.

5. "Which is the equivalent processing of human brain in terms of computer processing?". Accesso 4 dicembre 2020. <https://cs.stackexchange.com/questions/20016/which-is-the-equivalent-processing-of-human-brain-in-terms-of-computer-processing/40075>.

6. "Neural waves of brain." Accesso 4 dicembre 2020. <https://biophilic.blogspot.com/2011/05/neural-waves-of-brain.html>.

Sistemi multicore

Benché il multithreading sia in circolazione dalla fine degli anni Sessanta del secolo scorso⁷, l'interesse nei suoi confronti sta rapidamente aumentando in seguito alla proliferazione di sistemi multicore. Gli smartphone e i tablet sono comunemente dotati di processori multicore.

La prima CPU multicore venne introdotta da IBM nel 2001.⁸ La maggior parte dei nuovi processori oggi è almeno dual-core, sebbene i triple-, quad- e octa-core si stiano diffondendo sempre di più. Apple ha di recente presentato il processore M1 con 8 core nella CPU e fino a 8 ulteriori core nella GPU (*Graphics Processing Unit*).⁹ I processori di AMD per computer desktop arrivano fino a 32 core.¹⁰ Intel dispone di processori che hanno fino a 18 core¹¹ destinati al pubblico dei consumatori, e processori di fascia alta con fino a 72 core per supercomputer, server di fascia alta e sistemi desktop ad altissime prestazioni.¹² Per trarre pieno vantaggio dall'architettura multicore bisogna scrivere applicazioni multithreaded.

Complessità della programmazione concorrente

La scrittura di programmi multithreaded può risultare complicata. Benché la mente umana sia in grado di eseguire più funzioni contemporaneamente, è difficile riuscire a saltare tra flussi di pensiero paralleli. Per capire quanto sia impegnativo scrivere e comprendere programmi multithreaded, tentate il seguente esperimento: apriete tre libri diversi alla pagina 1 e cercate di leggerli contemporaneamente. Leggete alcune parole dal primo libro, poi alcune parole dal secondo e poi alcune dal terzo, quindi tornate indietro e leggete alcune parole successive dal primo libro, e così via. Dopo questo esperimento, vi renderete conto delle numerose sfide poste dal multithreading. Dovete:

- *passare* da un libro all'altro;
- *leggere* rapidamente;
- *ricordare a che punto siete* in ogni libro;
- *avvicinare il libro da leggere* in modo da poterlo vedere;
- *mettere da parte i libri che non state leggendo*.

Nel caos generato dalla veloce ripetizione di queste attività, dovete inoltre cercare di *comprendere* il contenuto dei libri!

Implementazione multithreading standard

In precedenza, le librerie multithreading del C erano estensioni non standard, dipendenti dalla piattaforma specifica. I programmati in C vogliono spesso che il loro codice sia portabile verso altre piattaforme. Questo è un vantaggio fondamentale del multithreading standardizzato. Il file di intestazione `<threads.h>` dichiara le nuove funzionalità (opzionali) per il multithreading per scrivere codice multithreaded più portabile.

Il Visual C++ di Microsoft e la versione di Apple del compilatore Clang in Xcode non supportano `<threads.h>`. Perciò abbiamo testato gli esempi di questo paragrafo usando:

- GNU gcc 10.2 su Ubuntu Linux;
- GNU gcc 10.2 nel contenitore Docker della GNU Compiler Collection, che può essere eseguito su Windows, macOS e Linux;
- GNU gcc 10.2 su Ubuntu Linux eseguito nel sottosistema Windows per Linux (WSL);
- Clang 10.0 su Linux.

7. "Thread (computing)" Accesso 4 dicembre 2020. [https://en.wikipedia.org/wiki/Thread_\(computing\)](https://en.wikipedia.org/wiki/Thread_(computing)).

8. "Power 4: The First Multi-Core, 1GHz Processor". Accesso 4 dicembre 2020. <https://www.ibm.com/ibm/history/ibm100/us/en/icons/power4/>.

9. "Apple unleashes M1". Accesso 18 novembre 2020. <https://www.apple.com/newsroom/2020/11/apple-unleashes-m1>.

10. "AMD unveils world's most powerful desktop CPUs". Accesso 18 novembre 2020. <https://www.zdnet.com/article/amd-unveils-worlds-most-powerful-desktop-cpus/>.

11. "IntelCoreProcessorFamily". Accesso 18 novembre 2020. <https://www.intel.com/content/www/us/en/products/processors/core.html>.

12. "Xeon Phi". Accesso 18 novembre 2020. https://en.wikipedia.org/wiki/Xeon_Phi.

In questa sezione presentiamo le funzionalità di base che vi permettono di creare ed eseguire thread e semplici applicazioni multithreaded. Alla fine della sezione introdurremo diverse altre funzionalità del multithreading che potete approfondire se siete interessati a creare applicazioni multithreaded più sofisticate.

Esecuzione di programmi multithreaded

Quando un programma è in esecuzione, le sue attività competono per l'assegnazione del processore con:

- il sistema operativo;
- altri programmi;
- altre attività che il sistema operativo esegue per vostro conto.

Quando eseguirete gli esempi di questo paragrafo, il tempo necessario per ogni calcolo varierà a seconda di:

- velocità del processore del computer;
- numero di core del processore;
- cosa è in esecuzione al momento sul computer.

È come andare in auto al supermercato. Il tempo che occorre per arrivarci può variare a seconda delle condizioni del traffico, del tempo atmosferico e di altri fattori. Certi giorni potreste impiegarci 10 minuti, ma durante le ore di punta o col tempo cattivo potreste metterci di più.

 Vi è anche un sovraccarico (*overhead*) intrinseco nel multithreading stesso. La semplice suddivisione di un'attività in due thread e la loro esecuzione su un sistema dual-core *non* porta a un'esecuzione due volte più veloce, sebbene sia normalmente più veloce rispetto all'esecuzione delle attività dei thread in sequenza. Eseguire un'applicazione multithreaded su un processore single-core può richiedere in realtà più tempo che eseguire semplicemente le attività dei thread in sequenza.

Panoramica degli esempi di questa sezione

Per dare una dimostrazione convincente dell'efficacia del multithreading su un sistema multicore, questa sezione presenta due programmi:

- uno effettua due calcoli intensivi *in sequenza*;
- l'altro esegue gli stessi calcoli intensivi con *thread paralleli*.

Gli output mostrati sono stati ottenuti usando il contenitore Docker della GNU Compiler Collection. Docker consente di specificare il numero di core dedicati al contenitore, al momento dell'avvio dello stesso, con l'argomento della riga di comando:

`--cpus=numeroDiCore`

Abbiamo eseguito ciascun programma utilizzando il contenitore Docker prima con un core e poi con due core, per verificare le prestazioni del programma nei due diversi scenari. Mostriamo per ciascun programma il tempo dei singoli calcoli e il tempo di calcolo totale. Gli output evidenziano i miglioramenti in termini di tempo quando il programma multithreaded viene eseguito su due core anziché su un singolo core.

C.9.1 Esempio: esecuzione sequenziale di due attività di calcolo intensivo

Le righe 35-42 della Figura C.5 definiscono la funzione ricorsiva `fibonacci`, analizzata originariamente nel Paragrafo 5.15. Come abbiamo visto in quel paragrafo, per valori di Fibonacci molto grandi, l'implementazione ricorsiva può richiedere un tempo di calcolo significativo. Questo esempio fa eseguire sequenzialmente i calcoli `fibonacci(50)` (riga 14) e `fibonacci(49)` (riga 23).

```

1 // figC_05.c
2 // Calcolo dei numeri di Fibonacci con esecuzione sequenziale
3 #include <stdio.h>
4 #include <time.h>
5

```

```

6 long long int fibonacci(int n); // prototipo di funzione
7
8 int main(void) {
9     puts("Sequential calls to fibonacci(50) and fibonacci(49)");
10
11    // calcola il valore di Fibonacci per 50
12    time_t startTime1 = time(NULL);
13    puts("Calculating fibonacci(50)");
14    long long int result1 = fibonacci(50);
15    time_t endTime1 = time(NULL);
16
17    printf("fibonacci(50) = %llu\n", result1);
18    printf("Calculation time = %f minutes\n\n",
19           difftime(endTime1, startTime1) / 60.0);
20
21    time_t startTime2 = time(NULL);
22    puts("Calculating fibonacci(49)");
23    long long int result2 = fibonacci(49);
24    time_t endTime2 = time(NULL);
25
26    printf("fibonacci(49) = %llu\n", result2);
27    printf("Calculation time = %f minutes\n\n",
28           difftime(endTime2, startTime2) / 60.0);
29
30    printf("Total calculation time = %f minutes\n",
31           difftime(endTime2, startTime1) / 60.0);
32 }
33
34 // Calcolo ricorsivo dei numeri di Fibonacci
35 long long int fibonacci(int n) {
36     if (0 == n || 1 == n) { // caso di base
37         return n;
38     }
39     else { // passo di ricorsione
40         return fibonacci(n - 1) + fibonacci(n - 2);
41     }
42 }
```

a) Esecuzione su un contenitore Docker con un core

```

Sequential calls to fibonacci(50) and fibonacci(49)
Calculating fibonacci(50)
fibonacci(50) = 12586269025
Calculation time = 1.700000 minutes

Calculating fibonacci(49)
fibonacci(49) = 7778742049
Calculation time = 1.050000 minutes

Total calculation time = 2.750000 minutes

```

b) Esecuzione su un contenitore Docker con due core

```

Sequential calls to fibonacci(50) and fibonacci(49)
Calculating fibonacci(50)
fibonacci(50) = 12586269025
Calculation time = 1.666667 minutes

Calculating fibonacci(49)
fibonacci(49) = 7778742049
Calculation time = 1.066667 minutes

Total calculation time = 2.733333 minutes

```

Figura C.5 Calcolo dei numeri di Fibonacci con esecuzione sequenziale.

Prima e dopo ogni chiamata di `fibonacci`, misuriamo il tempo corrente in modo da poter determinare il tempo totale richiesto per il calcolo. Usiamo poi questi dati per calcolare il tempo totale richiesto per entrambi i calcoli. Le righe 19, 28 e 31 usano la funzione `difftime` (dal file di intestazione `<time.h>`) per calcolare il numero di secondi di differenza tra i due tempi.

Il primo output mostra i risultati dell'esecuzione del programma nel contenitore Docker della GNU Compiler Collection con l'utilizzo di un core. Il secondo output mostra i risultati dell'esecuzione del programma con la configurazione del contenitore Docker impostata per usare due core. Il programma della Figura C.5 non usa il multithreading, quindi può essere eseguito solo su un singolo core, anche sul contenitore Docker che ne ha due. Nei nostri test, abbiamo eseguito più volte i programmi con uno e due core, e i risultati ottenuti ogni volta sono leggermente differenti. In generale l'uso di un singolo core ha richiesto più tempo, perché il processore doveva essere condiviso dal programma e dal Docker.

C.9.2 Esempio: esecuzione multithreaded di due attività di calcolo intensivo

Anche il programma della Figura C.6 usa la funzione ricorsiva `fibonacci` ma esegue ogni chiamata in un *thread separato*. Per compilare questo programma con GNU `gcc` (in Linux o nel contenitore Docker della GNU Compiler Collection) utilizzate il comando:

```
gcc -std=c18 figC_06.c -pthread
```

Il linker usa l'opzione `-pthread` per collegare il nostro programma alla libreria di threading del sistema operativo Linux. Se avete Clang su Linux, potete compilare il programma con:

```
clang -std=c18 figC_06.c -pthread
```

```

1 // figC_06.c
2 // Calcolo dei numeri di Fibonacci con thread separati
3 #include <stdio.h>
4 #include <threads.h>
5 #include <time.h>
6
7 #define NUMBER_OF_THREADS 2
8
9 int startFibonacci(void *nPtr);
10 long long int fibonacci(int n);
11
12 typedef struct ThreadData {
13     time_t startTime; // inizio dell'esecuzione del thread
14     time_t endTime; // fine dell'esecuzione del thread

```

```
15     int number; // numero di Fibonacci da calcolare
16 } ThreadData; // fine di struct ThreadData
17
18 int main(void) {
19     // dati passati ai thread: uso di inizializzatori designati
20     ThreadData data[NODES] =
21         {[0] = {.number = 50},
22          [1] = {.number = 49}};
23
24     // ogni thread richiede un identificatore di thread di tipo thrd_t
25     thrd_t threads[NODES];
26
27     puts("fibonacci(50) and fibonacci(49) in separate threads");
28
29     // crea e fai partire i thread
30     for (size_t i = 0; i < NODES; ++i) {
31         printf("Starting thread to calculate fibonacci(%d)\n",
32                data[i].number);
33
34         // crea un thread e controlla se la sua creazione ha avuto successo
35         if (thrd_create(&threads[i], startFibonacci, &data[i]) !=
36             thrd_success) {
37             puts("Failed to create thread");
38         }
39     }
40
41     // attendi che ogni calcolo sia completato
42     for (size_t i = 0; i < NODES; ++i) {
43         thrd_join(threads[i], NULL);
44     }
45
46     // determina il tempo in cui parte il primo thread
47     time_t startTime = (data[0].startTime < data[1].startTime) ?
48         data[0].startTime : data[1].startTime;
49
50     // determina il tempo in cui termina l'ultimo thread
51     time_t endTime = (data[0].endTime > data[1].endTime) ?
52         data[0].endTime : data[1].endTime;
53
54     // stampa il tempo totale di calcolo
55     printf("Total calculation time = %f minutes\n",
56            difftime(endTime, startTime) / 60.0);
57 }
58
59 // Funzione chiamata da un thread per il calcolo ricorsivo di Fibonacci
60 int startFibonacci(void *ptr) {
61     // cast di ptr a ThreadData * per accedere agli argomenti
62     ThreadData *dataPtr = (ThreadData *) ptr;
63
64     dataPtr->startTime = time(NULL); // tempo prima del calcolo
65
66     printf("Calculating fibonacci(%d)\n", dataPtr->number);
```

```

67     printf("fibonacci(%d) = %lld\n",
68         dataPtr->number, fibonacci(dataPtr->number));
69
70     dataPtr->endTime = time(NULL); // tempo dopo il calcolo
71
72     printf("Calculation time = %f minutes\n\n",
73         difftime(dataPtr->endTime, dataPtr->startTime) / 60.0);
74     return thrd_success;
75 }
76
77 // Calcolo ricorsivo dei numeri di Fibonacci
78 long long int fibonacci(int n) {
79     if (0 == n || 1 == n) { // caso di base
80         return n;
81     }
82     else { // passo di ricorsione
83         return fibonacci(n - 1) + fibonacci(n - 2);
84     }
85 }
```

a) Esecuzione su un contenitore Docker con due core

```

fibonacci(50) and fibonacci(49) in separate threads
Starting thread to calculate fibonacci(50)
Starting thread to calculate fibonacci(49)
Calculating fibonacci(50)
Calculating fibonacci(49)
fibonacci(49) = 7778742049
Calculation time = 1.083333 minutes

fibonacci(50) = 12586269025
Calculation time = 1.733333 minutes

Total calculation time = 1.733333 minutes
```

b) Esecuzione su un contenitore Docker con due core

```

fibonacci(50) and fibonacci(49) in separate threads
Starting thread to calculate fibonacci(50)
Starting thread to calculate fibonacci(49)
Calculating fibonacci(50)
Calculating fibonacci(49)
fibonacci(49) = 7778742049
Calculation time = 1.033333 minutes

fibonacci(50) = 12586269025
Calculation time = 1.600000 minutes

Total calculation time = 1.600000 minutes
```

c) Esecuzione su un contenitore Docker con un core

```

fibonacci(50) and fibonacci(49) in separate threads
Starting thread to calculate fibonacci(50)
Starting thread to calculate fibonacci(49)
Calculating fibonacci(50)
Calculating fibonacci(49)
fibonacci(49) = 7778742049
Calculation time = 2.150000 minutes

fibonacci(50) = 12586269025
Calculation time = 2.816667 minutes

Total calculation time = 2.816667 minutes

```

d) Esecuzione su un contenitore Docker con un core

```

fibonacci(50) and fibonacci(49) in separate threads
Starting thread to calculate fibonacci(50)
Starting thread to calculate fibonacci(49)
Calculating fibonacci(50)
Calculating fibonacci(49)
fibonacci(49) = 7778742049
Calculation time = 2.166667 minutes

fibonacci(50) = 12586269025
Calculation time = 2.833333 minutes

Total calculation time = 2.833333 minutes

```

Figura C.6 Calcolo dei numeri di Fibonacci con thread separati.

I primi due output mostrano l'esempio multithreaded di Fibonacci eseguito su un contenitore Docker con due core. Sebbene i tempi d'esecuzione siano stati diversi, il tempo complessivo per eseguire entrambi i calcoli di Fibonacci (nei nostri test) è stato sempre inferiore all'esecuzione sequenziale della Figura C.5: il tempo totale di esecuzione è stato lo stesso del calcolo di fibonacci(50) più lungo. La divisione del programma in due thread ha consentito che i due calcoli di Fibonacci venissero eseguiti simultaneamente, uno su ciascun core. Gli ultimi due output mostrano l'esempio eseguito su un contenitore Docker con un singolo core. Anche in questo caso, per ogni esecuzione i tempi sono stati diversi, ma il tempo complessivo è stato *maggior*e di quello dell'esecuzione sequenziale nella Figura C.5 per via del sovraccarico dovuto alla condivisione di *un solo* processore tra i thread del programma e il Docker.

struct ThreadData

Le righe 12-16 definiscono un ThreadData di tipo struct che contiene il numero da passare alla funzione fibonacci e due membri time_t dove memorizziamo il tempo prima e dopo ogni chiamata a fibonacci nei thread. La funzione che ogni thread esegue in questo esempio riceve un oggetto ThreadData come suo argomento. Le righe 20-22 creano un array ThreadData e usano inizializzatori designati (introdotti nel Paragrafo C.3) per impostare i loro membri number a 50 e 49 (i numeri di Fibonacci che calcoleremo).

thrd_t

La riga 25 crea un array di oggetti thrd_t. Quando si crea un thread, la libreria per il multithreading crea un *thread ID* (identificatore) unico e lo memorizza in un oggetto thrd_t. L'identificatore ID del thread può allora essere usato in varie funzioni di multithreading.

Creare ed eseguire un thread

Le righe 30-39 creano due thread chiamando la funzione `thrd_create` (riga 35). I tre argomenti della funzione sono:

- Un puntatore `thrd_t` che `thrd_create` usa per memorizzare l'ID del thread.
- Un puntatore a una funzione (`startFibonacci`) che specifica l'elaborazione da eseguire nel thread. La funzione deve restituire un `int` e ricevere un puntatore `void *` che rappresenta l'argomento della funzione. Il valore `int` di ritorno rappresenta lo stato del thread quando questo termina. Il puntatore `void *` consente a questa funzione di ricevere un argomento di qualsiasi tipo che sia appropriato per la vostra applicazione, nel nostro caso un puntatore a un oggetto `ThreadData`. Ricordiamo che qualsiasi tipo di puntatore può essere assegnato a un `void *`.
- Un puntatore `void *` all'argomento che `thrd_create` passerà alla funzione nel secondo argomento.

La funzione `thrd_create` restituisce `thrd_success` se il thread è creato, `thrd_nomem` se non c'è sufficiente memoria per allocare il thread, o altrimenti `thrd_error`. Se il thread è creato con successo, la funzione specificata come secondo argomento di `thrd_create` viene eseguita nel nuovo thread.

Congiunzione (join) dei thread

Per assicurarsi che il programma non termini finché non terminano i thread, le righe 42-44 chiamano `thrd_join` per ogni thread. Questo fa sì che il programma *aspetti* finché entrambi i thread non terminano prima di eseguire il codice rimanente in `main`. La funzione `thrd_join` riceve l'ID di `thrd_t` del thread da ricongiungere e un puntatore a un `int` dove `thrd_join` memorizza lo stato che il thread restituisce quando termina; se non avete bisogno di questo stato, passate `NULL` per questo argomento.

Calcolo dei tempi di esecuzione

Una volta che i thread sono terminati, le righe 47-56 calcolano e stampano il tempo totale di esecuzione determinando la differenza di tempo tra l'inizio del primo thread e la fine del secondo.

La funzione `startFibonacci`

La funzione `startFibonacci` (righe 60-75) specifica il compito da eseguire. In questo caso:

- chiamare `fibonacci` per eseguire ricorsivamente il calcolo;
- cronometrare il calcolo;
- stampare il risultato del calcolo;
- stampare il tempo che il calcolo ha richiesto (come abbiamo fatto nella Figura C.5).

Il thread viene eseguito finché `startFibonacci` non restituisce lo stato del thread (`thrd_success`, riga 74), e a quel punto il thread termina. Al termine dell'esecuzione di questa funzione termina anche il thread corrispondente.

C.9.3 Altre caratteristiche del multithreading

Esistono molte altre caratteristiche del multithreading, incluse variabili `_Atomic` e operazioni atomiche, memoria locale dei thread, condizioni e mutex (costrutti per la mutua esclusione). Per maggiori informazioni su questi argomenti consultate i Paragrafi 6.7.2.4, 6.7.3, 7.17 e 7.26 dello standard C18, i seguenti post di blog:

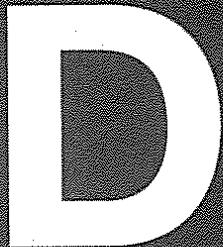
<https://smartbear.com/blog/test-and-monitor/c11-a-new-c-standard-aiming-at-safer-programming/>

e l'articolo:

<http://lwn.net/Articles/508220/>

Per la documentazione, consultate la pagine relative ai thread all'indirizzo:

<https://en.cppreference.com/w/c/thread>



Introduzione ai principi della programmazione orientata agli oggetti

D.1 Introduzione

È molto probabile che dopo aver appreso il C studierete uno o più linguaggi di programmazione orientata agli oggetti basati sul C o influenzati dal C, tra cui Java, C++, C#, Objective-C, Python, Swift e molti altri. Questi linguaggi spesso supportano alcuni paradigmi di programmazione:

- programmazione procedurale;
- programmazione orientata agli oggetti;
- programmazione generica;
- programmazione funzionale.

Questa appendice presenta una semplice panoramica della terminologia e dei concetti della programmazione orientata agli oggetti.

D.2 Linguaggi di programmazione orientata agli oggetti

Dal linguaggio C è derivata un'intera nuova generazione di linguaggi di programmazione che sono andati oltre il modello di programmazione procedurale del C. Data la sempre crescente richiesta per software nuovi e più potenti, è importante essere in grado di costruire software in modo veloce, corretto ed economico. Gli oggetti, o più precisamente le **classi** di oggetti, sono essenzialmente componenti software **riutilizzabili**. Esistono oggetti di diverso tipo: date, orari, audio, video, automobili, persone ecc. Quasi ogni sostantivo può essere rappresentato in maniera ragionevole come un oggetto software in termini di **attributi** (per esempio, nome, colore e dimensioni) e **comportamenti** (per esempio, calcolare, spostare e comunicare). I gruppi di sviluppo software possono essere molto più produttivi nella progettazione e nell'implementazione utilizzando un approccio modulare e orientato agli oggetti, piuttosto che le tecniche comunemente usate in precedenza. I programmi orientati agli oggetti sono spesso più semplici da capire, correggere e modificare.

D.3 L'automobile come oggetto¹

Per capire gli oggetti e i loro contenuti, considerate una semplice analogia. Supponete di voler guidare una macchina e farla andare più veloce premendo sul pedale dell'acceleratore. Che cosa deve accadere prima di poterlo fare? Innanzitutto, prima che voi possiate guidare una macchina, qualcuno deve progettarla. Normalmente si parte da disegni di ingegneria, simili alle planimetrie che descrivono il progetto di una casa. I disegni includono la progettazione del pedale dell'acceleratore. Questo pedale “nasconde” al guidatore il complesso meccanismo che fa andare la macchina più veloce, così come il pedale del freno nasconde il meccanismo che la fa rallentare, e il volante nasconde quello che la fa svoltare. Ciò permette che anche coloro che abbiano solo una minima o nessuna conoscenza del funzionamento del motore e dei meccanismi di frenata e di manovra siano in grado di guidare un’auto.

1. Mentre leggete il seguito di questa appendice, pensate a come le macchine a guida autonoma possano influire sulla discussione.

Così come non si possono cucinare pasti nella planimetria di una cucina, allo stesso modo non si può guidare il progetto di una macchina. Prima di poterla guidare, un'auto deve essere costruita partendo dal progetto che la descrive. Una macchina completa avrà un reale pedale dell'acceleratore, ma non è ancora sufficiente, infatti non accelererà da sola (si spera!): il guidatore dovrà premere il pedale.

D.4 Metodi e classi

Per eseguire un compito in un programma orientato agli oggetti è necessario un **metodo**. I metodi contengono le istruzioni del programma che eseguono i loro compiti. Ogni metodo nasconde queste istruzioni a chi lo utilizza, così come il pedale dell'acceleratore nasconde al guidatore i meccanismi di accelerazione. Nella programmazione orientata agli oggetti, una porzione di programma chiamata **classe** ospita un insieme di metodi che svolgono i compiti di quella classe. Per esempio, una classe che rappresenta un conto corrente bancario può contenere un metodo per effettuare versamenti, un altro per prelevare e un terzo per informare sul saldo del conto. Una classe è un concetto simile ai disegni di progettazione di un'automobile, che contengono il design del pedale dell'acceleratore, del volante e così via.

D.5 Istanziazione

Così come qualcuno deve costruire un'auto a partire dal progetto prima che altri possano guidarla, allo stesso modo qualcuno dovrà costruire un oggetto di una classe prima che un programma possa eseguire i compiti definiti dai metodi della classe. Questo processo è detto **istanziazione**, e l'oggetto è chiamato **istanza** della sua classe.

D.6 Riutilizzo

Così come il progetto di un'automobile può essere riutilizzato più volte per costruire molte macchine, allo stesso modo si potrà riutilizzare una classe per costruire molti oggetti. Il riutilizzo di classi esistenti quando si sviluppano nuove classi e programmi consente di risparmiare tempo e fatica. Il riutilizzo aiuta anche a costruire sistemi più affidabili ed efficaci. Le classi e i componenti esistenti spesso hanno già superato un'ampia fase di verifica (testing), debugging (trovare e correggere gli errori) e ottimizzazione. Così come il concetto di parti intercambiabili è stato cruciale nella Rivoluzione Industriale, le classi riutilizzabili sono essenziali nella rivoluzione software favorita dalla tecnologia a oggetti.

Nei linguaggi orientati agli oggetti, come C++, Java, C#, Python, Swift e molti altri, tipicamente si utilizza un **approccio a blocchi** per la creazione dei programmi. Per evitare di reinventare la ruota, useremo componenti di alto livello già esistenti ogni volta che ci sarà possibile. Questo riutilizzo di software è uno dei vantaggi fondamentali della programmazione orientata agli oggetti.

D.7 Messaggi e chiamate di metodo

Quando si guida un'auto, la pressione dell'acceleratore equivale all'invio di un messaggio all'auto affinché esegua un compito, in questo caso accelerare. Analogamente, si possono inviare messaggi a un oggetto. Ogni messaggio è implementato come una **chiamata di metodo** che indica al metodo di un oggetto di eseguire il suo compito. Per esempio, un programma può chiamare il metodo *deposito* di un oggetto conto corrente bancario per aumentarne il saldo di un dato importo.

D.8 Attributi e variabili di istanza

Un'automobile, oltre ad avere le capacità per eseguire determinati compiti, ha anche attributi, quali il colore, il numero di porte, la quantità di carburante nel serbatoio, la velocità attuale e i chilometri percorsi (cioè la cifra riportata dal contachilometri). Come le sue capacità, gli attributi di una macchina sono rappresentati come componenti del suo progetto e dei suoi schemi ingegneristici (che includono, per esempio, un contachilometri

e un indicatore per il carburante). Mentre si guida, questi attributi rimangono parte dell'auto stessa. Ogni auto gestisce solo i suoi attributi, per esempio ha le informazioni sul proprio livello di carburante ma non su quello delle altre macchine.

In modo simile, un oggetto possiede attributi che ne rimangono parte quando viene utilizzato in un programma. Questi attributi sono specificati come parte della classe dell'oggetto. Per esempio, un oggetto conto corrente avrà un *attributo saldo* che rappresenta l'importo di denaro presente nel conto. Ogni oggetto conto corrente conosce il saldo del conto che rappresenta, ma non i saldi degli altri conti della banca. Gli attributi vengono specificati dalle **variabili di istanza** della classe. Gli attributi e i metodi di una classe (e dei suoi oggetti) sono strettamente correlati, quindi le classi raccolgono e tengono assieme i propri attributi e metodi.

D.9 Ereditarietà

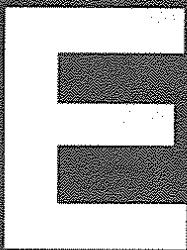
Si può creare una nuova classe di oggetti in maniera efficace per **ereditarietà**: la nuova classe (chiamata **sottoclasse**) inizierà con le caratteristiche di una classe già esistente (chiamata **superclasse**), con la possibilità di modificarle o di aggiungere nuove caratteristiche uniche. Nella nostra analogia con le auto, un oggetto di una classe "decappottabile" è sicuramente un oggetto della classe più generale "automobile", ma con una particolarità in più: il tetto si può alzare o abbassare.

D.10 Analisi e progettazione orientate agli oggetti (OOAD)

Molti programmatore creano il codice (ovvero, le istruzioni del programma) per i loro programmi senza una prima fase di progettazione. Questo approccio può funzionare per piccoli programmi come quelli presentati nei primi capitoli di questo libro. Ma cosa fareste se vi chiedessero di creare un sistema software per controllare migliaia di sportelli automatici di una banca? O supponete che vi venga chiesto di lavorare in un team di 1.000 sviluppatori di software per costruire la prossima generazione del sistema di controllo del traffico aereo statunitense?

Per creare la soluzione migliore in caso di progetti così vasti e complessi, dovete seguire un processo di **analisi** dettagliato per determinare i **requisiti** del vostro progetto (ovvero, definire che *cosa* deve fare il sistema). Dopodiché svilupperete un **progetto** che soddisfi questi requisiti (specificando *come* il sistema dovrà fare). Prima di scrivere il codice, l'ideale sarebbe svolgere tutto questo processo e sottoporre il progetto a un'attenta revisione, effettuata anche con l'aiuto di altri professionisti. Quando questo processo comporta l'**analisi** e la **progettazione** di un sistema da un punto di vista orientato agli oggetti, viene definito un **processo OOAD** (*Object-Oriented Analysis-and-Design*). La programmazione in un linguaggio orientato agli oggetti è chiamata **programmazione orientata agli oggetti (OOP, Object-Oriented Programming)** e permette di implementare un progetto orientato agli oggetti.

APPENDICE

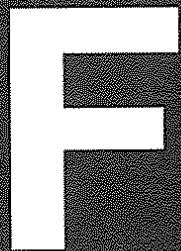


Sistemi di numerazione



L'Appendice E è disponibile sulla piattaforma Pearson MyLab

APPENDICE



Utilizzo del debugger Visual Studio

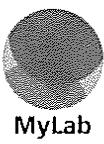


L'Appendice F è disponibile sulla piattaforma Pearson MyLab

APPENDICE

G

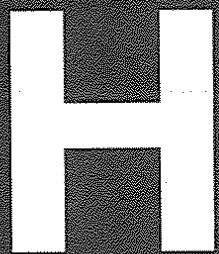
Utilizzo del debugger GNU



L'Appendice G è disponibile sulla piattaforma Pearson MyLab

MyLab

APPENDICE



Utilizzo del debugger di Xcode



L'Appendice H è disponibile sulla piattaforma Pearson MyLab

Indice analitico

[Nota: i numeri in grassetto indicano le pagine con la definizione dei termini.]

Simboli

\t (sequenza di escape di tabulazione orizzontale) 47
. (operatore virgola) 243
! (NOT, operatore di negazione logica) 131, 132
!= (operatore di uguaglianza) 57
? 77
? : (operatore condizionale) 77, 95, 189
. (operatore punto) 427
.wbt, file di Webots 328
* (carattere di soppressione dell'assegnazione) 414
* (operatore di moltiplicazione) 53, 88
* (operatori dei puntatori) 273
*= (operatore di assegnazione di moltiplicazione) 95
/ (operatore di divisione) 88
/*...*/ (commenti multilinea) 46

&= (operatore di assegnazione AND bit a bit) 443
(flag) 407
(operatore del preprocessore) 46, 606
(operatore del preprocessore) 606
% (carattere in una specifica di conversione) 51, 88, 396
% (operatore di resto) 53, 172
%% (specificia di conversione) 401
%= (operatore di assegnazione di resto) 95
%c (specificia di conversione) 164, 222, 401, 410
%d (specificia di conversione) 51, 52, 164
%E (specificia di conversione) 399, 411
%e (specificia di conversione) 399, 411
%f (specificia di conversione) 88, 164

- <=< (operatore di assegnazione di spostamento a sinistra) 443
 = (operatore di assegnazione) 51, 95
 - (operatore unario meno) 95
 -- (operatore di decremento) 93, 95, 291
 -= (operatore di assegnazione di sottrazione) 95
 -> (operatore di puntatore a struttura) 427
 > (operatore maggiore di) 57
 >> (operatore spostamento a destra) 436, 441
 >>= (operatore di assegnazione di spostamento a destra) 443
 | (operatore OR inclusivo bit a bit) 436
 |= (operatore di assegnazione OR inclusivo bit a bit) 443
 || (operatore logico OR) 131, 189
 ~ (operatore complemento) 436, 441
- A**
- a, modalità di apertura dei file 480
 a.out 18
 a+, modalità di apertura dei file 480
 ab, modalità di apertura dei file 480
 ab+, modalità di apertura dei file 480
 abort, funzione 608
 accelerometri 4
 accento circonflesso (^) 413
 accumulatore 319, 320, 323
 acronimi “as-a-Service”
 BDaaS (Big Data as a Service) 31
 Haas (Hadoop as a Service) 31
 IaaS (Infrastructure as a Service) 31
 PaaS (Platform as a Service) 31
 SaaS (Software as a Service) 31
 SaaS (Storage as a Service) 31
 affidabilità del credito 37
 affinamento graduale, 83, 297
 aggancio (*hook*, compilatore Simple) 567
 aggiungere un intero a un puntatore 290
 agglomerati di computer 39
 alberi 271, 424, 546
 della scena (Webots) 331, 332
 di ricerca binaria 546, 549, 550, 558
 figli 546
 figlio a destra 546
 figlio a sinistra 546
 fratelli 546
 nodo padre 546
 sottoalbero sinistro 546
 alberi binari XXIX, 546
 creare e attraversare 547
 profondità 558
 Alexa di Amazon XXXIII, 42
 algebra 53
- algoritmi 71
 completi 74
 di forza bruta 39
 di mescolamento imparziale 432
 di ordinamento per selezione 580, 582, 583
 ordinamento per fusione 587
 ordinamento per selezione 579
 ordinamento per inserzione 583
 per la valutazione dell'espressione postfissa 560
 sveglia 71
 sviluppo XXVII, XXX
 _Alignas, parola chiave 645
 aligned_alloc 645
 allarme (\a) 47
 allineare 396
 a destra in un campo 121, 396, 402, 406
 a sinistra 121, 396
 a sinistra in un campo 405
 stringhe in un campo 405
 allocare memoria 169
 allocazione dinamica prezzi vendita 37
 AlphaGo 40
 AlphaZero 40
 alternanza di bit 436
 ALU (*Arithmetic-Logic Unit*) 4
 ambiente di sviluppo
 integrato (IDE) 17
 per la programmazione in C 16
 ambiente locale 170
 AMD, processori 647
 American National Standards Committee per i computer e l'elaborazione delle informazioni 13
 American National Standards Institute (ANSI) 13
 analisi 657
 dati di un sondaggio XXVII, 232
 grafico sociale 37
 marketing 37
 risultati dell'esame 92
 sentiment 37, 507
 sondaggio di studenti, programma 218
 testi 378
 uso di array 232
 analizzabilità 635
 AND 436
 AND bit a bit (&), operatore 436, 440, 457
 Android
 sistema operativo 11
 smartphone 11
 animazione
 frame-by-frame 461
 in raylib 459
 Annex K XXXIII, 252
 rimozione dal C standard XXXIV
 annidamento 89, 90

- di blocchi costituenti 135
- ANSI** 13
- Apache Software Foundation 10
- Apple** 10
 - Macintosh 11
 - Siri XXXIII, 42
 - TV 11
 - Watch 11
 - Xcode XXVII, XLV
- Apple M1, processore 647
- applicazioni
 - client 517
 - di visione artificiale 39
 - mobile 11
 - multithread XXXVIII
- apprendimento con rinforzo 40
- approccio a blocchi 14, **656**
- arc4random**, funzione 192
- area di un cerchio 111
- argc** **616**
- argomenti 47, 50, **602**
 - della riga di comando XXIX, 467, 521, 616
 - di una funzione 156
 - liste di lunghezza variabile XXIX
- argv** **616**
- aritmetica 19
 - espressioni 290
 - media 55
 - operatori XXVII, **53**
 - operatori di assegnazione 93
 - operazioni 319
 - overflow **96**
 - regole di conversione **163**
- ARPANET 30
- array **212**
 - bidimensionali **242**, 245, 297, 298
 - controllo dei confini 219, 252
 - di caratteri 221, 222
 - di interi 212
 - di lunghezza variabile XXVII, **249**
 - di puntatori **296**, 305
 - di puntatori a funzioni 305
 - di stringhe **296**
 - dinamici **624**
 - inizializzatori **215**
 - JSON 518
 - m-per-n, **243**
 - notazione 293
 - notazione con indice 222, 283, 293
 - statici automaticamente inizializzati a zero 224
 - struttura di dati XXVII
- array multidimensionali XXVII, **242**, 243, 245
 - inizializzare 243
- array unidimensionali 286
- problema 264
- arresto di un programma 342, 400
- arrotondare **396**
 - nella direzione dell'infinito negativo 641
 - nella direzione dello zero 641
 - numeri 121
 - valori **88**
- ASCII (American Standard Code for Information Interchange)** **124**, 359
 - insieme di caratteri 6, 359
- assegnazione
 - espressioni 290
 - istruzione **51**
 - operatore = **51**, 57
- assembler 8
 - assert**, macro **608**
 - <**assert.h**> 169, **608**
 - asserzioni XXIX
 - statiche 635
 - Assistente Google XXXIII, 42
 - assistenti intelligenti XXXIII, 37, 42
 - Alexa di Amazon XXXIII, 42
 - Assistente Google XXXIII, 42
 - Cortana di Microsoft XXXIII, 42
 - Siri di Apple XXXIII, 42
 - Watson di IBM XXXIII, 42
 - assistenti personali 37
 - assistenti virtuali intelligenti **508**
 - assistenza disabili 37
 - associatività **54**
 - degli operatori 212, 275, 443
 - asterischi iniziali 379
 - asterisco (*) **53**
 - astrazione **158**
 - at_quick_exit** **644**
 - atexit**, funzione **619**
 - _Atomic**, variabile **654**
 - attività
 - eseguite concorrentemente **646**
 - eseguite in parallelo **646**
 - attraversamento
 - di un albero binario **547**
 - di un albero binario per livelli successivi **558**
 - di un labirinto 191, 316
 - in ordine di un albero binario 191, **547**, 550
 - in post-ordine **547**, 550
 - in post-ordine di un albero binario 191
 - in pre-ordine di un albero binario 191, **547**, 550
 - attributi **656**
 - di un oggetto **656**
 - di una classe **655**
 - auto**, specificatore della classe di memoria **178**
 - auto a guida autonoma 37, 39
 - autovalutazioni XXX

avanzamento pagina, carattere (\f) 344
 Awesome C, elenchi di librerie 14
 azione/decisione, modello 76
 azioni 47, 57, 71, 72

B

B, linguaggio 12
 backslash (\) 47, 408, 603
 base 8, sistema numerico 349
 base 10, sistema numerico 349
 base 16, sistema numerico 350
 BASIC, linguaggio di programmazione 15
 BCPL 12
 BCryptGenRandom, funzione 192
 BDaaS (Big data as a Service) 31
 Bell Laboratories 12
 big data XXIII, XXVII, 7, 36
 analisi 36
 bit (“cifra binaria”) 4, 6
 del segno 397
 di ordine più alto 437
 di ordine più basso 437
 Bitcoin 35, 43, 387
 Bjarne Stroustrup 15
 blocchi costituenti accatastati 138
 blocco 46, 78, 161
 campo d’azione 181
 di controllo del file 476
 di dati 365
 esterno 180, 181, 183
 interno 180
 blockchain 35, 43
 Böhm e Jacopini 73
 bool XXIX, 635, 636
 _Bool, tipo di dati 133, 638
 booleano, tipo 133, 638
 break 125, 129, 130, 153
 Brick, gioco (esercizio) 473
 bubble sort 230, 232, 261, 284, 286, 302
 con passaggio per riferimento 284
 bucket sort 595
 buffer fuori schermo 461
 buone pratiche di programmazione XL
 byte 4, 6, 435

C

C
 community open source XXXII
 documentazione standard XXX
 Language Reference (Microsoft) XLI
 linguaggio 12
 preprocessore 17, 46, 599
 repository di codice XXXII

standard ISO/IEC 9899:2018 13
 C++, linguaggio di programmazione 15, 163
 C#, linguaggio di programmazione 16
 C11 XXX, 635
 C11, file di intestazione 643
 C18 XXX, 27, 635
 C99 635
 C99, file di intestazione 636
 CAI (*computer-assisted instruction*) 208
 calcoli 4, 52, 62
 California Consumer Privacy Act (CCPA) XXXIII, 42
 calloc 624
 campo 6
 d’azione di un identificatore 180, 603
 di bit 444
 di bit anonimo 446
 di bit anonimo con larghezza zero 447
 Webots 331
 Cannon, gioco (caso pratico sulla programmazione di giochi) 463
 miglioramenti 472
 capacità di calcolo parallelo 38
 carattere di escape 47, 408
 carattere nullo ('\0') 221, 222, 281, 296, 342, 562
 di terminazione 221, 342, 352, 400
 caratteri 6
 di controllo 346
 e stringhe (specificatore di conversione) 400
 letterali 396
 per la scansione 410
 caratteri di spaziatura 46, 75
 per separare stringhe letterali 232
 caratteri speciali 341
 set 6
 caricare
 un programma in memoria 319
 valori 573
 caricare/mettere in memoria operazioni 319
 Carnegie Mellon University, Software Engineering Institute XXXVIII, 644
 cartelle sanitarie elettroniche 37
 case, etichette 125, 126, 180
 casi
 d’uso 37
 di base 183
 pratici XXIII
 cast 604
 cbrt, funzione 157
 CCPA (*California Consumer Privacy Act*) XXXIII, 42
 ceil, funzione 158
 Celsius 421
 CERT

- Divisione del Software Engineering Institute della Carnegie Mellon University XXXVIII, 43, 644
 standard di codifica per il C 43, 307
- `char`, tipo di dati 164
- `char`, tipo primitivo 342
- `char * 400`
- `char ** 348`
- `CHAR_BIT`, costante simbolica 438
- `chatbot 508`
- chiamare una funzione 156, 160
- chiamata 160
- chiave di ricerca 237
- chiave privata 338, 388, 390
 per decriptografare 388
- chiave pubblica 388, 388, 389
- chiave pubblica/chiave privata, coppia 388
- chiavi di ordinamento 577
- ciclo 80, 83, 87, 115
 controllato da contatore 89, 90
 di esecuzione delle istruzioni 322, 323
 di gioco 461
 infinito 80, 87, 117
- cifra binaria (bit) 6
- cifrari
 a chiave segreta di Vigenère XXXV, 382, 383, 386, 387
 a sostituzione 382
 algoritmi 382
`cryptii.com` 382
 di Cesare 382, 435
- cifre 69
- cima di una pila 523
- `cimag`, funzione 639
- circonferenza di un cerchio 111
- cJSON, libreria 517, 520
- Clang, contenitore XLV, 2, 20
- `clang-tidy` XXXIII
- classe di memoria 179
 di un identificatore 179
- classi 655
 in linguaggi di programmazione orientati agli oggetti 186
 variabili di istanza 656
- clienti
 abbandono 38
 agenti di servizio 37
 fidelizzazione 38
- Climate at a Glance, serie temporali 517
- `CloseWindow`, funzione (raylib) 462
- cloud XXIII, XXXVII, 30, 517
 computing 30
 servizi XXXVII, 30, 518
- code XXIX, 271, 424, 523, 540
- sistemi di elaborazione 540
- codice 1
 approccio “dal vivo” XXIII
 esempi XL
 legacy 280
 numerico 359
 oggetto 17
 operativo 319, 566
 operativo non valido 323
 ottimizzato 573
 portabile 14
 repository di XXXII
 sorgente 9
- codifica, standard XXXIV
- coercizione degli argomenti 163
- collegamento 178, 523
 di un identificatore 178
 esterno 618, 641
 interno 618, 641
- colonne 242
- `Color`, tipo in raylib 461
- colori in raylib 459
- combinazione di 1 e 0 6
- Command Line Tool, progetto in Xcode 24
- commenti 46
- commissioni 107, 261
- Communications of the ACM* 73
- compilare ed eseguire un progetto in Xcode 25
- compilatori 9, 17, 46, 47
 Clang 2, 20
 GNU gcc XLV, 2, 20
 ottimizzazione 572
- Visual Studio Community Edition 2, 20
- Visual Studio di Microsoft XLV
- Xcode su macOS XLV, 2, 20
- compilazione 16, 17
 condizionale XXIX, 599, 604
 di programmi con più file sorgente XXIX
- complemento
`(~)`, operatore 436
 a uno 441
 bit a bit `(~)`, operatore 438, 441
- complessità esponenziale 189
- `complex`, numero 639
`_Complex`, parola chiave 640
`complex.h` 636
`<complex.h>`, file di intestazione, 157
- componenti 655
 software riutilizzabili 655
- comportamenti
 di una classe 655
 indefiniti 415, 644
- compromesso spazio-tempo 592

- computer
 - hardware XXIII
 - nell'esperienza formativa 208
 - quantistici 35
 - software XXIII
- Computer Science and Artificial Intelligence Laboratory (CSAIL) XXXI
- concatenare stringhe 355, 378
- concorrenza 646
- condividere
 - memoria (unioni) 433
 - veicoli 38
- condizione 57, 131
 - di continuazione del ciclo 113, 114, 115, 116, 127
 - semplice 131
- confine
 - di parola 426
 - di unità di memoria 447
- confini, controllo dei 219, 252
- confrontare
 - file, programma 503
 - stringhe 355
- confronto lessicografico di stringhe 359
- conoscenza dei propri dati 511, 514
- const, parola chiave 229, 280, 282, 296
- contatori 81, 108
 - di dati (compilatore Simple) 570
 - smart 38
- conteggio
 - dei cicli 115
 - della frequenza delle parole 508
 - di voti a lettera 122
 - statistiche 512
- contentore Docker XLVII
- conti correnti bancari, programma 492
 - esempio, 119
- conti creditori 149
- continue 129, 130, 153
- controllare se una stringa è palindroma 190
- controller (Webots) 334, 335, 337, 338
- controllo
 - dei tipi 163
 - del programma 72
 - dell'allineamento in memoria 635
 - dell'intervallo 141
 - inventario 38
 - ortografico 508
 - sugli errori (nell'elaborazione dei file) 491
 - traffico 38
- convalidare dati 141
- convenzioni di questo libro per font del testo XL
- conversione
 - da infisso a postfisso 560
 - esplicita 87
- implicita 87
- metrica, programma 380
- convertire lettere minuscole in lettere maiuscole 169
- convertitore da infisso a postfisso, esercizio 560
- copia 171
 - temporanea 87
- copiare stringhe 355, 378
- coppia di parentesi più interne 54
- coprimo 389
- corpo
 - di un while 80
 - di un'istruzione 59
 - di una funzione 46
- corpus/corpora 507
- correggere 73
- correzione ortografica 508
- correzioni 19
- cos, funzione 158
- coseno 158
- costanti 560
 - carattere 281, 341
 - di enumerazione 177, 447, 604
 - simboliche 125, 216, 599
 - simboliche predefinite 607
- costruire un compilatore XXX, 523, 559, 563, 565, 568, 570, 571, 572, 574
- costruire un computer 321
 - caso pratico XXX, XXXV, XXXVII, 8, 523
- CPU (*Central Processing Unit*) 4
- Craigslist 31
- Craps, gioco 175, 208
 - statistiche 263
- “crashing” 85
- creat, funzione 639
- creare frasi 376
- crescita della popolazione mondiale 153
 - esercizio 112
- crimine, sorveglianza predittiva 38
- criptovalute 35, 43, 387
- crittografare 112
- crittografia XXXV, 382
 - a chiave pubblica XXXV, 387, 388, 389
- crittografia simmetrica 387
 - testo in chiaro 382
- cronometraggio, operazioni di XXXIII
- crowdsourcing dei dati, 38
- cruciverba, generatore di 381
- Cryptography API: Next Generation di Microsoft 192
- CSAIL (Computer Science and Artificial Intelligence Laboratory) XXXI
- .csv, estensione di nomi di file 515
- CSV, file di testo con valori separati da virgole XXVIII

<Ctrl> c 621
 <ctype.h>, file di intestazione 169, 603
 Cyberbotics Ltd. 324

D

data mining 7
 data munging 510
 data science XXVII, XXVII, XXVIII, 32, 512
 casi d'uso 37
 conoscenza dei propri dati 512
 progetto di visualizzazione dinamica delle statistiche di vincita/perdita del gioco Craps 473
 visualizzazione dinamica del lancio di due dadi 472
 visualizzazione dinamica del lancio di una moneta 472
 data wrangling 510
 database 7
 relazionale 7
 dataset 512
 esempi 513
 date 169
 DATE, costante simbolica predefinita 607
 dati 2
 esempi 513
 gerarchia XXVI, 5
 tipi principali XXVII
 deallocate memoria 525
 Debug, area (Xcode) 24
 debugging XXX, 656
 decisioni 57, 62
 logiche 2
 decrementare un puntatore 290
 decremento 114, 117, 290
 decrittografare 112
 deep learning 39, 42
 DeepBlue di IBM 39
 default, caso 125, 126
 #define, direttiva per il preprocessore 216, 601
 definizioni 50
 deque 540, 541
 dereferenziare un puntatore a void * 292
 descrittore di file 476
 design pattern 32
 determinare la lunghezza delle stringhe 355
 deviazione standard 512
 di operatori aritmetici 59
 diagnosi/trattamento tumori 38
 diagnostica 169
 diagnosticare condizioni mediche 39
 diagramma di flusso 73, 75
 dell'istruzione do...while 128

per struttura sequenziale 73
 strutturato 137
 diametro di un cerchio 111
 difftime, funzione (file di intestazione time.h) 650
 direttive per il preprocessore 17, 599, 600, 603
 disco 18
 a stato solido 3, 4
 fisso 3, 17
 disegnare grafici 150
 dispositivi 17, 19
 di input 3
 di memoria persistenti 5
 di output 4
 distanza tra punti 206
 distruttivo 52, 53
 dividere per zero 323
 dividi e conquista 155, 158
 divisione intera 54, 87
 affidabile 635, 641
 divisioni 4, 53
 per zero 19, 85
 dizionario 506
 do...while
 esempio di istruzioni 127
 istruzioni di iterazione 74
 Docker XLVII
 contenitore della GNU Compiler Collection (GCC) XLVII, 2, 20, 27, 28, 648
 Desktop, installer XLVII
 immagini XLVII
 Docker Hub, account XLVII
 documentare i programmi 46
 doppie virgolette ("") 47
 DOS (*Disk Operating System*) 10
 (double), operatore cast 87
 double, tipo 86, 87, 120, 163
 double complex 639
 DrawGame, funzione (raylib) 462
 DrawRectangleLines, funzione (raylib) 471
 DrawTextureEx, funzione (raylib) 471
 dump 322
 duplicati, eliminazione di 262, 268, 550, 558

E

Eclipse Foundation 10
 École polytechnique fédérale de Lausanne (EPFL) 325
 economia collaborativa 38
 editing 16, 19
 editor 16, 341
 Editor, area (Xcode) 24
 EEP (esempi, esercizi e progetti) XXXI, XXXII

- effetti secondari 170, 188
 efficienza
 ordinamento per fusione 592
 ordinamento per inserzione 586
 ordinamento per selezione 583
 elaborazione
 del linguaggio naturale (NLP) XXVIII, XXXI, XXXVI, XXV, 38, 507, 508
 del testo 341
 di file, controllo sugli errori 491
 di transazioni, programma 492
 interattiva 51
 elementi di un array 212
 al di fuori dei confini 252
 elenchi separati da virgole 243
 elevamento a potenza modulare 392
 elevare al cubo una variabile
 usando il passaggio per riferimento 276
 usando il passaggio per valore 276
 elevare un intero a una potenza intera 190
#elif 604
 ellissi (...) nel prototipo di funzione 613
#else 604
 else sospeso, problema 79, 109, 110
 emacs 17
 e-mail (posta elettronica) 30
 Empty Project, modello 20
#endif 604
 end-of-file 123, 342, 351, 475, 478
 enqueue 541
 enum 177, 447
 enumerazioni XXVIII, 177, 448
 esempio 448
 EOF 125, 343
 e-puck_avoid_obstacles, controller (Webots)
 334
 e-reader, dispositivo 12
 ereditarietà 657
`<errno.h>` 169
#error, direttiva per il preprocessore 606
 errore 19
 condizioni 169
 di compilazione 17, 134
 di rappresentazione nei numeri in virgola mobile
 121
 di segmentazione 51, 342, 400
 di sintassi 17, 79, 95, 96, 135
 di tipo off-by-one 117
 in fase di compilazione 17
 in fase di esecuzione 19
 irreversibile 19, 54, 69, 85, 323, 401
 logico 79, 83, 117, 134, 216, 433
 logico irreversibile 79
 messaggi 19
 non irreversibile 19, 54, 69, 79, 163
 esadecimali 151, 343, 349, 396, 397
 esecuzione sequenziale 73
 eseguire 18
 fase 18
 in modo condizionale le direttive per
 il preprocessore 599
 in parallelo XXXVII
 un programma 3
 esempi, esercizi e progetti (EEP) XXXI, XXXII
 espandere una macro 602
 esponenti 55
 modulari 392
 espressioni 122, 127, 161
 con tipi misti 164
 condizionali 77
 costanti integrali 127
 di confronto 290
 di controllo 125
 di tipo generico 635
 estrazione di frasi nominali 508
 Ethereum 35, 43
 etica XXXIII, 43
 etichette 180, 625
 Eulero 265
 euristica 267
 di accessibilità 266
 eventi di input in raylib 459
 evento 621
 exabyte (EB) 33
 exaflop 35
exit, funzione 619
 `atexit`, funzione 619
EXIT_FAILURE 619, 643
EXIT_SUCCESS 619, 643
 exp, funzione 157
extern 179, 617
- F**
- f o F per un float 621
fabs, funzione 157
 Facebook 10
 Fahrenheit, temperature 421
false, condizione 57, 636
 fare direttamente/indirettamente riferimento
 a un valore 272
 fattore di scala 172, 175
 fattoriali 112, 149
 FCB 476
fclose, funzione 478
fenv.h 636
feof, funzione 478, 491

- fgetc, funzione 476, 506
 fgets, funzione 351, 476
 Fibonacci, funzione 189
 Fibonacci, serie di 187, 205
 FIFO (first-in first-out) 540
 file 6, 475
 - ad accesso casuale 485, 488
 - ad accesso diretto XXVIII, XXXVI
 - ad accesso sequenziale 476
 - binari XXVIII, 479
 - campo d'azione 180
 - delle transazioni 503
 - di testo XXVIII
 - nomi 17
 - per la memorizzazione dei dati a lungo termine XXVIII
 - principali 503
 FILE_, costante simbolica predefinita 607
 FILE, puntatore 476
 file di intestazione 46, 133, 169, 600
 - <complex.h> 157, 636, 639
 - <ctype.h> 343
 - di input/output standard (stdio.h) 46
 - di intestazione per argomenti variabili stdarg.h 614
 - <fenv.h> 636
 - includere nei programmi 169
 - <inttypes.h> 636
 - personalizzati 170
 - <stdbool.h> 636, 638
 - <stdint.h> 636
 - <stdio.h> 351
 - <stdlib.h> 348
 - <string.h> 355
 - <tgmath.h> 636
 filtrare modelli di progetti in Visual Studio 20
 filtro anti-spam 381
 "fine dell'inserimento dei dati" 83
 fine della coda (tail) 523, 540
 finestra dello spazio di lavoro in Xcode 24
 finestra principale in Visual Studio 21
 first-in first-out (FIFO) 540
 Fisher-Yates, algoritmo di mescolamento 432
 fisica in Webots 339
 flag 396, 405
 - spazio 406
 flipped classroom XXXIX
 float, tipo di dati 121, 164,
 <float.h> 169
 floor, funzione 158
 FLOPS (*floating-point operations per second*) 34
 flusso di controllo 62
 fmod, funzione 158
 Folding@home, rete 34
 fopen, funzione 478
 for, istruzione di iterazione 74, 117
 forma lineare 54
 formato
 - esponenziale 396, 397
 - tabellare 214
 formattazione XXVIII
 forme in raylib 459
 forzare la privacy con la crittografia 112
 fprintf, funzione 476
 fprintf_s, funzione 498
 fputc, funzione 476
 fputs, funzione 476
 frame al secondo 462
 fread, funzione 476, 486
 free, funzione 525, 552
 fscanf, funzione 476
 fscanf_s, funzione 498
 fseek, funzione 488
 - func_, identificatore predefinito 643
 funzioni XXVII, 14, 17, 46, 140, 156
 - argomenti 156
 - campo d'azione 180
 - chiamanti 156
 - chiamata di 156, 160
 - chiamata e ritorno 170
 - corpo 161
 - della libreria math 169, 207
 - della Libreria Standard del C 14
 - di confronto di stringhe 357
 - di conversione di stringhe 348
 - di Fibonacci 190
 - di gestione della memoria della libreria per il trattamento delle stringhe 365
 - di utilità 169
 - esponenziali 157
 - fattoriali 185, 190
 - intestazione 160, 304, 306
 - invocate 156, 160
 - iterative 239
 - meccanismo di chiamata/ritorno XXVIII
 - nome 160, 180, 302
 - non ricorsive 205
 - parametro 160, 277, 283
 - per la manipolazione di stringhe della libreria per il trattamento delle stringhe 355, 359
 - per la ricerca della libreria per il trattamento delle stringhe 359
 - personalizzate XXVII
 - pila delle chiamate XXVII
 - predicato 533
 - prototipi 120, 160, 163, 180, 277, 286

puntatore 302, 305
 restituire valori 156, 157
 toziente di Eulero 389
 trigonometriche coseno 158
 trigonometriche seno 158
 trigonometriche tangente 158
 fusione di due array 587
`fwrite` 476, 486, 488

G

gara tra la tartaruga e la lepre 209
 rappresentazione multimediale con raylib 468
 Gates, William XXXII
`gcc`, comando per la compilazione 18
`GDPR (General Data Protection Regulation)`
 XXXIII, 43
 generatore di parole per numeri telefonici 505
 generazione di numeri casuali 297, 376
`_Generic`, parola chiave 644
 gestione dinamica della memoria 271, 525
`getc` 603
`getchar` 352, 506, 603
 gigabyte (GB) 4, 33
 gigaflop 34
 giochi 38, 171
 “Craps” 175, 263
 da casinò XXVII, XXXIV
 di carte 313
 di dadi 175
 programmazione XXXI, XXXV
 sistemi XXXIII
 Giro del Cavallo 265
 approcci a forza bruta 267
 test di chiusura del giro 268
`GitHub` XXXI, XXXII, 10
 GNU
`gcc` XXVII, XLI, XLV, 2, 20
 Libreria Scientifica 512, 515
 manuale di riferimento della Libreria Standard
 del C XLI
`gnuplot` XXXVII, 512
 installazione 515
 Go, gioco da tavolo 40
 Google Maps 31
 Gosling, James 15
`goto`, istruzione 73, 180, 624
 eliminazione 73
`GPS (Global Positioning System)`, dispositivo 3
 sensore 38
`GPU (Graphics Processing Unit)` 647
 grafici
 a barre 150, 219
 a tartaruga 264

gravità in Webots 328
`gsl_fit_linear`, funzione 515
`GUI (Graphical User Interface)` 11

H

Hadoop as a Service (HaaS) 31
`halt` 322
 hardware XXIII, XXVI, 1, 2, 8
`HIPAA (Health Insurance Portability and Accountability Act)` XXXIII, 42
 host del servizio web 517
`HTML (HyperText Markup Language)` 30
`HTTP (HyperText Transfer Protocol)` 30
`HTTPS`, protocollo 382

I

identificatori 50, 602
 identificazione della lingua 508
`#if` 604
`if`, istruzione di selezione 57
`if...else`
 istruzione di selezione 74, 77
 istruzioni annidate 77, 79
`#ifdef`, direttiva per il preprocessore 604
`#ifndef`, direttiva per il preprocessore 604
 immagini
 carte da gioco 460, 469
`Docker` XLVII
 eseguibili 18
 immunoterapie 37
`#include`, direttiva per il preprocessore 600
 incrementare
 puntatori 290, 291
 variabili di controllo 114, 118
 indentazione 75, 78
 indici 212, 219
 indirezione 272, 276
 doppia (un puntatore a un puntatore) 530
 indirizzi
 del puntatore 530
 di memoria 271
 di un campo di bit 446
 IP 30, 31
 “Indovina il numero”, esercizio 205
 informatica scientifica 15
 informazioni
 di movimento 4
 di posizione 4
 volatili 4
 Infrastructure as a Service (IaaS) 31
 ingegneria del software 130, 180, 286
 osservazioni XLI
`InitGame`, funzione in un gioco raylib 462

- InitWindow, funzione (raylib) **462**
- inizializzare **50**
 - array multidimensionali **243**
 - elementi di un array a zero **214**
 - elementi di un array con una lista di inizializzatori **215**
 - strutture **426**
- inizializzatori
 - designati **XXIX**, **635**, **636**, **637**, **653**
 - lista **221**, **243**
- inizio di una coda **523**
- inline**, funzione **635**, **643**
- input **XXVII**
 - inserire
 - caratteri letterali **396**
 - in un albero binario **191**
 - insieme di caratteri **69**, **124**, **341**
 - insieme di scansione **412**
 - invertito **413**
 - int**, tipo **46**, **50**, **164**
 - Intel, processori **647**
 - intelligenza artificiale (IA) **XXVII**, **XXVIII**, **39**
 - generale **39**, **42**
 - intercetta **514**
 - intercettare eventi **621**
 - interesse composto **119**, **120**, **149**
 - Interface Builder **11**
 - interi **46**, **50**
 - allineare a destra in un campo **402**
 - casuali con spostamento e variazione di scala
 - prodotti da **1 + rand() % 6** **171**
 - decimali con segno **397**
 - decimali senza segno **397**
 - esadecimali **274**
 - esadecimali senza segno **397**
 - intero costante **283**
 - ottali senza segno **397**
 - senza segno **435**
 - specifica di conversione **397**
 - International Standards Organization (ISO) **13**
 - Internet **30**, **517**
 - larghezza di banda **XXIII**
 - Internet delle cose (IOT, *Internet of Things*) **XXIII**, **11**, **31**, **36**, **38**, **43**
 - dispositivi medici **37**
 - interpreti **9**
 - interruzioni **621**
 - intervallo (statistiche) **512**
 - inttypes.h** **636**
 - inventario **505**
 - inviare messaggi a un oggetto **656**
 - Invio*, tasto **18**, **51**, **126**, **323**
 - invocare una funzione **156**, **160**
 - iOS **10**, **11**
 - IP (*Internet Protocol*) **30**
 - iPad **11**
 - iPadOS **11**
 - iPhone **11**
 - ipotenusa di un triangolo rettangolo **202**
 - isalnum**, funzione **343**
 - isalpha**, funzione **343**
 - isblank**, funzione **343**
 - iscntrl**, funzione **344**
 - isdigit**, funzione **343**
 - isgraph**, funzione **344**
 - islower**, funzione **343**
 - ISO (International Standards Organization) **13**
 - ISO/IEC 9899 2018 (documento del C standard) **13**
 - isprint**, funzione **344**, **346**
 - ispunct**, funzione **344**, **346**
 - isspace**, funzione **344**, **346**
 - Issue, navigatore **24**
 - istanze **656**
 - istanziazione **656**
 - istruzione assistita da computer (CAI) **208**
 - livelli di difficoltà **209**
 - monitorare le prestazioni dello studente **209**
 - ridurre l'affaticamento dello studente **208**
 - variare i tipi di problemi **209**
 - istruzioni **18**, **47**, **73**, **563**
 - composte **78**
 - contrassegnate (*flagged*) **566**
 - di controllo **XXVII**
 - di controllo a un solo ingresso e a una sola uscita **74**, **76**, **138**
 - di iterazione **XXVII**, **74**, **80**
 - di salto **323**
 - di salto non condizionato **572**, **625**
 - di selezione **XXVII**, **75**
 - di selezione doppia **74**, **90**
 - di selezione multipla **74**, **126**
 - di selezione singola **74**
 - vuote **59**
 - istruzioni *branch* **566**
 - negative **567**
 - zero **567**, **570**, **571**
 - istruzioni di controllo annidate **89**
 - accatastamento **74**, **75**
 - annidamento **74**
 - istruzioni *halt* **566**
 - contatore **570**
 - return** **160**
 - isupper**, funzione **343**
 - isxdigit**, funzione **343**
 - IT (*Information Technology*) **7**
 - iterazione **189**
 - controllata da contatore **XXVII**, **81**, **114**, **115**

controllata da sentinella XXVII, 84
definita 82
indefinita 83

J

Jacopini, G. 73
Java, linguaggio di programmazione 11, 15
JavaScript 16
Jobs, Steve 11
JSON (JavaScript Object Notation) XXXVII, 38, 518
 array 518
 false 518
 formato basato su testo per lo scambio di dati 518
 libreria cJSON 517, 521
 null 518
 numeri 518
 oggetti 518
 stringhe 518
 true 518
 valori booleani 518

K

Kasparov, Gary 39
kernel 10
 di un sistema operativo 10
Kernighan, B. W. 13
Kotlin, linguaggio di programmazione 11

L

l o L, suffisso per un letterale long double 521
l o L, suffisso per un letterale long int 621
labirinti
 di ogni dimensione 317
 generare in modo casuale 317
lancio di un dado 172, 175, 220
 simulazione XXXI
 usare array al posto di switch 220
lancio di una moneta 204
larghezza di banda 3, 30
larghezza di campo 121, 396, 402, 404, 413
 leggere dati 413
 larghezza di un campo di bit 444, 447
last-in, first-out (LIFO) 165, 534
latino maccheronico, esercizio 376
legge dei grandi numeri 465
Legge di Moore 3, 37
leggere caratteri e stringhe 412
leggere dati con una larghezza di campo 413
leggere input con interi (specificatori di conversione)
 410
leggere input in virgola mobile (specificatori di
 conversione) 410

Leggi di De Morgan 152
leggibilità 59, 507
letterali 47
 composti XXIX, 635, 637
 letterali in virgola mobile 121
lettere 6
 maiuscole 69, 169
 minuscole 6, 69, 169
lettura ed eliminazione di caratteri dallo stream
 di input 414
libcurl, libreria XXXVII, 517, 520
Libreria Standard del C 14, 16, 156, 171, 280
 file d'intestazione XLI
 funzioni XXVIII
libreria standard di input/output (stdio) 351
librerie
 di utilità generale (stdlib) 348
 per il trattamento dei caratteri 343
librerie standard 17
 file di intestazione 169, 600
LIFO (last-in, first-out) 165, 534
limerick, esercizio 376
limiti di credito 149
limiti per i valori interi 169
<limits.h>, file di intestazione 169, 438
__LINE__, costante simbolica predefinita 607
#line, direttiva per il preprocessore 607
linee di flusso 73
linguaggi
 assembly 8
 di alto livello 9

<locale.h>, file di intestazione 170
 locazioni 52
 log, funzione 157
 log_n confronti 551
 log10, funzione 157
 logaritmo naturale 157
 Logo, linguaggio 264
 long 127
 long double 121, 164
 long int 164
 long long int 164
 lvalue (“left value”) 134, 212

M

M1, processore (Apple) 647
 macchina virtuale XXVIII, XXXI, XXXV, 318, 559
 machine learning XXXI, XXXVII, 39, 512, 514
 Macintosh 11
 macOS 11
 macro 169, 599, 602
 argomenti 602
 complex 639
 definizione 602
 di tipo generico 642
 espansione 602
 identificatori 602
 lista di argomenti di lunghezza variabile 640
 poco sicure 608
 maggiore tra due numeri 105
 main 46
 main ricorsivo 190
 malloc, funzione 525, 624
 manipolazione di stringhe, esercizi avanzati 378
 mappatura cerebrale 37
 maschera 437
 mashup XXXVII, 31, 517
 massimo (statistiche) 512
 massimo comun divisore 190, 392
 “massimo livello di precedenza” 54
 math generico, tipo 635
 <math.h>, file di intestazione 120, 163, 169
 maximum 161
 funzione definita dal programmatore 187
 mazzo di carte 296
 media XXXIV, 236
 aritmetica 55, 236
 mediana XXXIV, 232, 236
 medicina
 di precisione 37
 diagnostica 37
 personalizzata 37
 preventiva 37

megabyte (MB) 32
 membri 424
 array flessibili XXIX, 635, 642
 di una struttura 424
 memchr, funzione 365, 367
 memcmp, funzione 365, 365
 memcpy, funzione 365, 365, 641
 memmove, funzione 365
 memoria 3, 4, 18
 automatica 179, 225
 concetti XXVII
 dinamica, allocazione XXIX, 624
 principale 4
 secondaria 3, 5, 17
 memorizzare valori 573
 memset, funzione 365, 368
 mescolare e distribuire le carte 297, 300, 430
 ad alte prestazioni 430
 messaggio 47
 messaggio di avviso o squillo 408
 metodo 656
 chiamate 656
 Microsoft XXXII
 Cortana XXXIII, 42
 Visual C++ XXVII
 Visual Studio Community Edition XLV, 20
 minimi quadrati, metodo dei 514
 minimizzazione rischi 37
 minimo (statistiche) 512
 minimo comune multiplo 393
 misure di dispersione 512
 deviazione standard 512
 varianza 512
 misure di tendenza centrale 512
 misure di variabilità 512
 MIT, Project MAC XXXI
 moda XXXIV, 232, 236, 262
 dati bimodali 262
 dati multimodali 262
 modalità
 di apertura del file 478, 479
 esclusiva di scrittura 480
 modelli
 di input/output formattato 485
 software 321
 modificatori di lunghezza 397
 moltiplicazione 53
 di due interi 191
 mondo in Webots 328
 motore fisico (Webots) 334
 mouse 2
 Mozilla Foundation 10
 multicore, computer XXXI

- multipli di un intero 111
 multithreading XXIX, XXXI, XXXIII, XXXVII,
 540, 646
 -pthread, opzione 650
 mutex (multithreading) 654
- N**
- “*n* fattoriale” (*n!*) 184
 Nadella, Satya XXXII
 National Oceanic and Atmospheric Administration
 (NOAA) 517
 Navigator, area (Xcode) 24
 Issue 24
 Project 24
 negazione logica 131, 132
 neutralità di genere 381
 NeXTSTEP, sistema operativo 11
 n-grammi 508
 NodeJS 16
 nodi 525, 526
 foglia 546
 radice di un albero binario 546, 558
 Webots 331
 nome 114, 212
 di un array 212
 di una variabile, 52
 di una variabile di controllo 114
 non distruttivo 53
 _Noreturn, specificatore di funzione 635, 644
 normali regole di conversione aritmetica 163
 notazione
 a cammello 50
 con indice 283
 esponenziale 399
 in virgola fissa 398
 infissa 560
 postfissa 560
 puntatore/indice 293
 puntatore/offset 293
 scientifica 399
 Notepad++, editor di testo 460
 NULL 272, 292, 478, 530
 puntatore 524, 624
 null in JSON 518
 numeri
 binari 151, 343
 casuali XXVII, XXXIV, 169
 complessi XXIX, 635, 636, 639
 di posizione 212
 di riga 563, 565
 in JSON 518
 numero maggiore di un insieme 108
 ottali 151, 343, 349, 396, 397
 perfetti 204
 primi 204, 392
 pseudocasuali 173
 telefonici, programma 377
 numeri casuali sicuri
 arc4random, funzione 192
 BCryptGenRandom, funzione 192
 random, funzione 192
 numeri decimali 152, 343, 349
 cifre 6
 nuova riga (*newline*, \n) 47, 222, 341, 342, 344, 414
- O**
- O grande, notazione XXIX, 578, 579, 583
 O(1) 578
 O(*n*), tempo 578
 O(*n* log *n*), tempo 592
 O(*n*²), tempo 578, 583, 586
 Objective-C 11
 occultamento delle informazioni 180, 286
 offset 292, 490
 di file 482
 oggetti 655
 attributi 656
 OOAD (*Object-Oriented Analysis-and-Design*) 657
 OOP (*Object-Oriented Programming*) 657
 open source 10, 11
 aumento della produttività 10
 codice XXXII
 community XXXII
 librerie 38
 movimento XXIII
 software XXXI, 38
 OpenAI 10
 OpenCV 10
 OpenML 10
 OpenWeatherMap XXXVII
 Current Weather Data 518
 One Call API 518
 servizio web 517
 operandi 51, 319, 566
 operatori XXVII, 93
 AND logico (&&) 131, 437
 binari 51
 bit a bit XXVIII, 435
 bit a bit AND, OR inclusivo, OR esclusivo e
 complemento 438
 bit a bit di spostamento 441
 cast 86, 87, 164
 condizionale (?) 77, 95
 di assegnazione =, +=, -=, *=, /= e %= 93

- di assegnazione bit a bit 443
 di assegnazione per l'addizione ($+=$) 93
 di decremento ($--$) 93
 di dereferenziazione (*) 274, 427, 428
 di incremento (++) 93
 di incremento e di decremento prefissi 94
 di indirezione (*) 274, 276
 di indirizzo (&) 51, 222, 273, 276, 286, 287
 di input/output 319
 di postdecremento 94
 di postincremento 94
 di postincremento o di postdecremento postfissi 94
 di predecremento 94
 di preincremento 94
 di spostamento a destra ($>>$) 456
 di spostamento a sinistra ($<<$) 436, 456
 di uguaglianza ($==$) 57
 di uguaglianza e relazionali 292
 esponenziale, 119
 freccia ($->$) 427
 logici XXVII, 131
 moltiplicativi 88
 NOT logico (!) 131, 132
 OR logico (||) 131
 punto (.) 427
 ternario 77, 189
 relazionali 57, 67
 unario 87, 95, 287
 virgola (,) 189, 243
 operazioni atomiche 654
 OR esclusivo bit a bit (^), operatore 436, 440
 OR inclusivo bit a bit (|), operatore 436, 440
 ordinamento 230, 577
 array XXVII
 con albero binario 550
 per fusione, algoritmo 587, 592
 per inserzione, algoritmo 583, 584, 586
 per selezione 191, 595
 per selezione ricorsivo 595
 ordinare algoritmi
 bucket sort 596
 per fusione 587
 per inserzione 583
 per selezione 580
 quicksort 596
 ordine 71, 73, 75
 di valutazione degli operandi 188
 orologio 174
 analogico, esercizio 473
 digitale, esercizio 473
 osservazioni in una serie temporale 516
 ottimizzazione 656
 compilatore Simple 573
 otto regine 191, 267, 269
 approcci a forza bruta 268
 output XXVII
 overflow 621
 del buffer 222, 252
 dell'accumulatore 323
- P**
- π 151
 pacchetti 30
 pagina logica 409
 palindromi 268
 paradigmi di programmazione XXXIV, 655
 parallelismo 646
 parametri di una funzione 160
 parentesi
 () 54, 59
 annidate 54, 55
 graffa destra, () 46
 graffa sinistra, () 46
 graffe {{}} 78
 quadre [] 212
 ridondanti 56
 parole 319
 riservate 59
 parole chiave 59
 aggiunte in C11 60
 aggiunte in C99 60
 parti frazionarie 87
 passare
 array e singoli elementi di array a funzioni 227
 per riferimento XXVIII, 226, 227, 271, 276, 277, 286, 429
 per valore 276, 278, 429
 passo di partizionamento di quicksort 596
 passo ricorsivo di quicksort 596
 passo temporale di base (Webots) 335
 pendenza 514
 percentuale, segno (%) 53
 permanenza 179, 180
 in memoria 178, 179, 225
 in memoria di un identificatore 179
 in memoria statica 179
 petabyte (PB) 33
 petaflop 2, 34
 phishing 506
 scanner 506
 piattaforme hardware 13
 PII (*personally identifiable information*) 43
 pile XXIX, 165, 271, 424, 523, 534
 delle chiamate delle funzioni 165, 282

- di esecuzione del programma 165
- programma 535
- Platform as a Service (PaaS) 31
- poker 313
- polinomi 55
- pop 535
- pop su una pila 165
- portabilità 14
- POS tagging (Parts-of-speech tagging) 508
- posizioni degli elementi della libreria per la programmazione dei giochi raylib 461
- posposizione indefinita 298, 314, 432
- potenza 158
- pow (power), funzione 55, 120, 158
- #pragma, direttiva per il preprocessore 606
- precedenza 54, 212, 275
- precedenza degli operatori 59, 212
 - regole 54
 - tabella 631
- precisione 88, 121, 396, 397, 399
 - per interi, numeri in virgola mobile e stringhe 403
 - predefinita 88, 399
- predizione
 - epidemie di malattie 37
 - risultati di salute 37
 - vendite basate sulle condizioni atmosferiche 38
- preelaborazione 16
- prelevamento (*fetch*) della successiva istruzione da eseguire 323
- prelevare 322
- preprocessore XXIX, 17, 170, 599
- prestazioni XLI, 14
- prevenzione
 - di attacchi terroristici 38
 - di epidemie di malattie 38
 - di furti 38
- previsioni
 - del mercato azionario 38
 - del tempo 38
- primo affinamento 84, 90
- principio del privilegio minimo 180, 229, 276, 280, 282, 286
- printf, funzione 47, 395
- printf_s, funzione XXXIII
- privacy XXXII, XXXV, 42, 382
- privilegi d'accesso 280, 283
- probabilità 171
- problemi
 - consumo del carburante 105
 - conversione di un numero binario in decimale 111
 - interesse semplice 107
 - limiti di credito 106
 - media di una classe 81, 86
- palindromi 111
- risoluzione XXVII, XXX
- risultati dell'esame 91
- stampare un quadrato di asterischi vuoto 110
- procedura 71
- processo di compilazione 570
- processori 2
 - dual-core 4
 - multicore XXXVII, 4
 - octa-core 4
 - quad-core 4
- product 67
- produzioni (compilatore Simple) 573
- progettazione, processo 657
- progetti
 - Gutenberg XXXII, 509
 - in Visual Studio 20
 - in Xcode 24
 - requisiti 657
- programma per simulare il gioco Craps 175-177
- ProgrammableWeb XXXVII, 31, 521
- programmare, fondamenti XXIII
- programmatori 2
- programmazione
 - di giochi, casi pratici 463, 464
 - di sistemi embedded, caso pratico XXXIV
 - errori comuni XL
 - funzionale XXXIV, 655
 - generica XXXIV, 655
 - orientata agli oggetti (OOP, *Object-Oriented Programming*) XXXIV, 11, 15, 158, 655, 657
- procedurale XXXIV, 655
- senza goto 73
- strutturata 45, 62, 71, 73, 135, 625
- Programmazione sicura in C, paragrafi XXXVIII
- programmi 2
 - autodocumentanti 50
 - con più file sorgente 179, 617, 618
 - di traduzione 8
 - eseguibili 47
 - fase di caricamento 323
 - fase di chiusura 85, 88
 - fase di elaborazione 83, 85, 88
 - fase di inizializzazione 85
 - multifunzione di ordinamento con puntatori a funzioni 302
 - terminazione anomala 621
- Project MAC (MIT) XXXI
- Project, navigatore 24
- promozione 164
 - intera 164
- prompt 50
 - dei comandi, finestra 20, 22

- proporzione aurea 187
 protezione di assegni 379
 PROTO, nodi (Webots) 332
 pseudocodice 72, 85, 88
 -pthread, opzione (multithreading) 650
 pubblico dominio
 immagini 470
 immagini di carte da gioco 460, 469
 puntatore XXVII, 271, 272, 275, 276
 a funzioni 302
 a struttura 427
 a un puntatore (indirezione doppia) 350, 530
 a void (void *) 291
 aritmetica 290, 291, 293, 378
 confronti 292
 costante 283, 283, 292
 costante a dati costanti 280, 283
 costante a dati non costanti 280, 283, 283
 di posizione del file 482
 espressione 292, 293
 generico 291
 indicizzazione 293
 non costante a dati costanti 280, 281
 non costante a dati non costanti 280
 notazione 277, 293, 293
 operatore freccia (->) 427
 parametro 276
 ristretto 635, 641
 “sospeso” 552
 variabile 283
 punto e virgola (;) 47, 59
 push 535, 538
 push su una pila 165
 putchar 351
 puts 351, 506
 funzione 61
 Python 15
 Python Software Foundation 10
- Q**
 quartetto di Anscombe XXVIII, XXXVII, 512, 514
 quattro V dei big data 36
 quick_exit, funzione 635
 quicksort 191, 596
- R**
 R, linguaggio di programmazione 15, 16
 r, modalità di apertura dei file 480
 r+, modalità di apertura dei file 480
 radianti 158
 radice cubica, funzione 157
 radice quadrata 157
- raggio 111
 raise 621
 RAM (*Random Access Memory*) 4
 rand 171
 RAND_MAX 171, 174
 random, funzione di sistemi basati su POSIX 192
 randomizzazione 173
 rappresentazione temporanea double 120
 raylib, guida rapida (*cheat sheet*) 461
 raylib, libreria per la programmazione dei giochi XXXV, 458
 animazione XXXVI, 458
 animazione frame-by-frame 461
 ciclo di gioco 461
 CloseWindow, funzione 462
 Color, tipo 461
 colori 459
 colori RGBA 461
 costanti di colore 461
 DrawGame, funzione 462
 DrawRectangleLines, funzione 471
 DrawTextureEx, funzione 471
 eventi con input XXXVI, 459
 forme XXXVI, 459
 giochi di esempio 459
 gioco Cannon 464
 InitGame, funzione 462
 InitWindow, funzione 462
 legge dei grandi numeri 465
 programmi demo in C 459
 Rectangle, tipo 461
 rFXGen, generatore di effetti sonori online 463
 rilevamento delle collisioni XXXVI, 459
 SetTargetFPS, funzione 462
 Sound, tipo 461
 suoni XXXVI, 459
 tipi 461
 tipi personalizzati 461
 UnloadGame, funzione 462
 UpdateGame, funzione 462
 Vector2, tipo 461
 WindowShouldClose, funzione 462
 raylib.h, file di intestazione 461
 rb, modalità di apertura dei file 480
 rb+, modalità di apertura dei file 480
 read 566
 readline, funzione (non standard) 343
 realloc 624
 realtà virtuale 324
 record 6, 282, 476
 chiave 476
 di attivazione 165

- Rectangle, tipo in raylib **461**
- RectangleArena (Webots) **327, 331**
- refactoring **32**
- register **178**
- regole
 - accatastamento **137**
 - “annidamento” **138**
 - conversione **163**
 - precedenza degli operatori **54**
- regressione lineare **514**
 - semplice XXVIII, XXXVII, **514**
- reindirizzare input o output **396**
- reinventare la ruota **156**
- relazione lineare **514**
- Representational State Transfer (REST) **518**
- requisiti **179**
 - di prestazioni **179**
- resa delle colture **38**
- RESTful, servizi web **518**
- restituire
 - da una funzione **156, 157**
 - un risultato **46, 160**
- resto **158**
 - operatore (%) **53, 68, 172**
- restrict **641**
- reti di computer **540**
- reti neurali **40**
- retta di regressione **514**
- rettangolo **76**
- return **275**
 - istruzione **160, 161**
- rewind, funzione **482**
- rFXGen, generatore di effetti sonori online (raylib) **463**
- RGBA (rosso, verde, blu, alpha), colori **461**
- riassunto
 - automatico **38**
 - di testi **508**
- ricerca
 - all'interno di stringhe **355**
 - binaria **19, 191, 237, 238, 239, 269**
 - in array, XXVII, **237**
 - in un albero binario **551**
 - in una lista collegata **191**
 - in una lista in modo ricorsivo **558**
 - lineare **191, 237, 268**
 - visuale prodotti **38**
- Richards, Martin **12**
- richiesta
 - a un servizio web **517**
 - di terminazione **621**
- riconoscimento
 - di entità nominate **508**
- facciale **38**
- vocale **37, 38, 508**
- ricorsione XXVII, **183, 189**
 - chiamata ricorsiva **183**
 - di coda **208**
 - funzione ricorsiva **183**
 - funzione ricorsiva gcd **206**
 - funzione ricorsiva power **205**
 - infinita **186**
 - passo di ricorsione **183**
 - rispetto all'iterazione **189**
- ricorsione, esempi
 - attraversamento di un labirinto **191**
 - attraversamento in ordine di un albero binario **191**
 - attraversamento in post-ordine di un albero binario **191**
 - attraversamento in pre-ordine di un albero binario **191**
 - controllare se una stringa è palindroma **190**
 - elevare un intero a una potenza intera **190**
 - funzione di Fibonacci **190**
 - funzione fattoriale **190**
 - inserimento in un albero binario **191**
 - main ricorsivo **191**
 - massimo comun divisore **190**
 - moltiplicare due interi **190**
 - ordinamento per selezione **191**
 - otto regine **191**
 - quicksort **191**
 - ricerca binaria **191**
 - ricerca in una lista collegata **191**
 - ricerca lineare **191**
 - somma degli elementi di un array **190**
 - stampare all'indietro una lista collegata **191**
 - stampare all'indietro una stringa in ingresso dalla tastiera **191**
 - stampare un array **190**
 - stampare una stringa all'indietro **190**
 - Torri di Hanoi **190**
 - valore minimo in un array **190**
 - visualizzare la ricorsione **190**
- riduzione inquinamento **37**
- riferimenti
 - in avanti (compilatore Simple) **567**
 - non risolti **617**
- righe **242**
- rilevamento
 - anomalie **37**
 - collisioni in raylib **459**
 - emozioni **37**
 - frodi **37**
 - malware **37**
 - similarità **37, 507, 509**

- spam 37
 trend 37
 ripetibilità 173
 riproducibilità XXXIX, 42
 risparmio di memoria 573
 risposta da un servizio web 517
 Ritchie, D. 12
 ritorno a capo ('\r') 392
 riusabilità del software 14, 158, 286, 618
 rivoluzione informatica 3
 robot
 e-puck (Webots) 327
 simulati XXXIV
 robotica, simulazioni XXXIV
 simulatore Webot XXXIV
Romeo and Juliet XXXVI
 RSA, algoritmo di crittografia a chiave pubblica XXXV, 388
 problema 394
 rompere un testo cifrato creato
 dall'implementazione RSA 394
 ruote differenziali (Webots) 327
rvalue ("right value") 134
- S**
- scacchi 39, 265
 scacchiera 111
 scalari 227, 286
 scambiare valori 579, 583
scanf, funzione 50, 395
scanf_s, funzione XXXIII, 252
 scansione di immagini 3
 schermo 2, 4, 19
 scomposizione 159
 scrivere su un file 478
SDK (Software Development Kit) 32
 secondo affinamento 84, 85, 91
Secure Coding in C and C++, 2/e 141
SEEK_CUR 490
SEEK_END 490
SEEK_SET 490
 segnale di attenzione interattivo 621
 segnali, gestione XXIX, 621
 libreria 621
 SEI (Software Engineering Institute della Carnegie Mellon University) 43
 SEI CERT, standard di codifica per il C XXIII, 43, 60
 seme 173
 per la funzione *rand* 173
 seno 158
 sensibilità all'uso di maiuscole/minuscole 50
 separare i caratteri con delimitatori 363
- sequenza di escape 47, 408, 421
 sequenziamento genoma umano 37
 serie temporali 516
 Climate at a Glance 517
 multivariate 516
 osservazioni 516
 univariate 516
 servizi basati sulla posizione 37
 servizi web XXXVII, 30, 517
 host del servizio web 517
 invocare con libcurl 520
 richiesta 517
 risposta 517
 Setaccio di Eratostene 268
<setjmp.h>, file di intestazione 170
SetTargetFPS, funzione (raylib) 462
 sezione aurea 187
 Shakespeare XXXVI, 509
 shell su Linux 2, 20
short 127, 164
 sicurezza XXXV, XXXVIII, 382
 vulnerabilità 61, 415
 sicurezza informatica (cybersecurity,) 37
SIGABRT 621
SIGFPE 621
SIGILL 621
SIGINT 621
SIGINT, intercettare 623
signal 621
<signal.h>, file di intestazione 169, 621
SIGSEGV 621
SIGTERM 621
 silicio 2
 simboli 69, 74
 connettori 74
 di azione 74
 di decisione 76
 rettangolo 74
 rettangolo arrotondato 74
 rombo 76
 speciali 6
 Simple, compilatore 565
 aggancio 567
 caso pratico, costruire un compilatore 565
 contatore di dati 570
 miglioramenti 574
 ottimizzazione 573
 primo passaggio 565
 produzioni 573
 riferimento in avanti irrisolto 567
 secondo passaggio 566
 tabella dei simboli del compilatore 565
 token 566

- Simple, linguaggio di programmazione inventato 559, 563
- Simpletron 506
computer, caso pratico XXXV
macchina virtuale 559, 563
simulatore 318, 321, 323, 324
- Simpletron Machine Language (SML) XXXV, XXXVII, 318, 524
istruzione 319
- simulatore
di computer 321
di robotica 323, 339
- simulazione XXXI, 171, 297
del lancio di dadi 459, 465
software XXXV, 318, 321
tecniche XXVII, XXXIV
- `sin`, funzione 158
- sinking sort 230
- sintesi vocale 38, 508
- sistema
di comunicazione XXXIII
di numerazione XXX
di prenotazione per compagnie aeree 263
di raccomandazione 38
embedded (incorporato) XXXIII, 3, 12, 13
guidato da menu 305
in tempo reale XXXIII
multicore XXXIII, 647
numerico binario 6, 397, 436
operativo XXXIII, 10, 13
prestazioni intensive XXXIII
- `size_t` 214, 355
- `sizeof`
applicazione al nome di un array per ottenere il numero di byte nell'array 287
operatore 287, 425, 506, 525, 604
- smart city 38
- smart home 38
- smartphone 11
- SML (Simpletron Machine Language) XXXV, XXXVII, 318, 319, 524
- SMS, linguaggio per 381
- SOA (*Service-Oriented Architecture*) 31
- software XXIII, XXVII, 1
componenti riutilizzabili 655
dei sistemi, casi pratici XXVI
per l'impaginazione 341
proprietario 10
- Software as a Service (SaaS) 31
- Software Development Kit (SDK) 32
- Software Engineering Institute (Carnegie Mellon) XXXVIII, 43, 644
- Solution Explorer 21
- aggiungere un file esistente 21
visualizzare 21
- soluzioni in Visual Studio 20
- sommare
due numeri 105
elementi di un array 191, 217
programma 49
- somme 4
statistiche 512
- sondaggio 217
- sottoalbero destro 546
- sottoclasse 657
- sottosistema Windows per Linux (WSL) XXVII, XLI, XLVI, 23
- sottotitolazione automatica 38
- sottrarre
un intero da un puntatore 290
un puntatore da un altro puntatore 290
- sottrazioni 4
- Sound, tipo in raylib 461
- sovraposizione di regioni in memoria 641
- spazi 75, 414
- specificatori della classe di memoria 179
- specificatori di conversione 396, 407
 `0` (zero), flag 407
 `c` 401
 `e` o `E` 398
 `f` 398
 `g` o `G` 398
 `s` 401
 `x` o `X` 397
specifiche di conversione 50, 51, 396
 `%c` 222
 `%d` 50, 51
 `%p` 274
 `%s` 61
 `%zu` 214
spostamento 172, 490
- SpotOn, caso pratico sulla programmazione di giochi 463
 miglioramenti del gioco 471
- `sprintf` 351
- `sqrt`, funzione 157
- `rand` 173
- `sscanf` 351, 354
- stack overflow 166
- StackOverflow XXXI, XXXII
- stampare
alberi 558
alberi binari 558
all'indietro una lista collegata 191
all'indietro una stringa in ingresso dalla tastiera 191

- array 190, 269
- caratteri 345, **346**
- date in vari formati 379
- interi senza segno nella loro rappresentazione in bit 436
- numeri positivi e negativi con e senza il flag + 406
- più righe con una singola `printf` 48
- quadrati 110
- righe con due istruzioni `printf` 48
- schemi 149
- stringhe all'indietro 190, 269
- stringhe un carattere alla volta usando un puntatore non costante a dati costanti 281
- valori di un'unione con entrambi i tipi di dati dei membri 434
- standard input 50
- static** 179, 180, 224
- static**, variabile globale 463
- _Static_assert** 608, **645**
- statistiche
 - conteggio 512
 - descrittive XXVIII, XXXIV, **512**, 513
 - deviazione standard 512
 - gioco Craps 263
 - intervallo 512
 - massimo 512
 - minimo 512
 - misure di dispersione 512
 - misure di tendenza centrale 512
 - misure di variabilità 512
 - somma 512
 - varianza 512
- statistiche descrittive di base XXVIII, XXXIV
 - media 236
 - mediana 236
 - moda 236
- `stdalign.h`, file di intestazione 645
- `stdarg.h`, file di intestazione 169, 614
- `stdbool.h` 133, 636, **638**
- _STDC_**, costante simbolica predefinita 607
- `<stddef.h>`, file di intestazione 170, 272
- `stderr` (stream standard error) **19**, **476**
- `stdin` (stream standard input) **19**, **351**, **476**
- `<stdint.h>` 636
- `<stdio.h>`, file di intestazione 46, 125, 169, 181, **351**, **395**, **476**, 603
- `<stdlib.h>`, file di intestazione 169, 171, **348**, 619
- `stdout` (stream standard output) **19**, **476**, **478**
- STEM (corsi universitari di scienza, tecnologia, ingegneria e matematica) XXIV
- stemming **508**
- “stop word”, eliminazione **508**
- Storage as a Service (SaaS) 31
- Store 320
- `strcat`, funzione **355**
- `strchr`, funzione **359**
- `strcmp`, funzione **357**
- `strcpy`, funzione **355**
- `strcspn`, funzione **361**
- stream **395**, **475**
- stream standard error (`stderr`) **19**, **395**, **476**
- stream standard input (`stdin`) XXVIII, **19**, **395**, **476**
- stream standard output, (`stdout`) XXVIII, **19**, **395**, **476**
- `strerror` **369**
- `<string.h>`, file di intestazione 169, 335
- stringhe **47**, **341**
 - come puntatori 342
 - costanti **341**
 - di caratteri 47, 213
 - di controllo del formato **51**, **396**, **403**, **409**
 - elaborazione XXVIII, 169, 221
 - immutabili (non modificabili) 342
 - letterali 221, **341**, 342
 - letterali separate solo da un carattere di spaziatura 232
 - non modificabili (immutabili) 342
 - per ricerche 359
 - terminate da un carattere nullo 296
- `strlen`, funzione **369**
- `strncat`, funzione **355**
- `strncmp`, funzione **357**
- `strncpy`, funzione **355**
- `strupr`, funzione **360**, **361**
- `strrchr`, funzione **360**, **362**
- `strspn`, funzione **362**, **363**
- `strstr`, funzione **360**, **363**
- `strtod`, funzione **348**
- `strtok`, funzione **360**, **363**
- `strtol`, funzione **349**
- `strtoul`, funzione **348**, **350**
- `struct` **211**, **424**
- strumenti di analisi statica del codice XXXIII
- strutture XXVIII, **282**, **424**
 - autoreferenziali **424**, **524**
 - definizione **424**, **425**
 - di controllo, 73, 74
 - di dati XXVIII, 523
 - di dati non lineari 546
 - di dati statiche **624**
 - di selezione 73, 74
 - dinamiche di dati XXVIII, 271, **523**
 - etichette **424**, **425**
 - lineari di dati **526**

- occupare lo spazio libero 446
- operatore di membro di struttura (.) 427, 428, 434
- operatore di puntatore a struttura (->) 427, 428, 434
- sequenziali 73, 74
- tipo 424
- suddividere
 - array nell'ordinamento per fusione 587
 - stringhe in token 355, 363, 377
- suffissi
 - per interi 621
 - per letterali in virgola mobile 621
- suffissi interi
 - l o L per un long int 620
 - ll o LL per un long long int 620
 - u o U per un unsigned int 620
- sum 67
- suoni in raylib 459
- superclasse 657
- supercomputer 2 35
- supermercato, simulazione di un 557
- sviluppare algoritmi XXVII
- Swift, linguaggio di programmazione 11, 16
- switch, istruzione di selezione multipla 74, 122, 126

- T**
- tabelle
 - dei file aperti 476
 - dei simboli (compilatore Simple) 565
 - di valori 242
 - di verità 131
- tablet 11
- tabulazione 47, 69, 75, 409, 414
 - orizzontale (\t) 47, 344
 - verticale ('\'v') 344
- tan 158
- tangente 158
- tastiera 2, 49, 50, 351
- tavola (o quadrato) di Vigenère 382, 386
- TCP (*Transmission Control Protocol*) 30
- TCP/IP 30
- telemedicina 38
- tempo 169
 - virtuale 335
- tempo di esecuzione
 - costante 578
 - lineare 578
 - per un algoritmo nel caso peggiore 577, 578
 - quadratico 579
- terabyte (TB) 5, 33
- teraflop 34
- terminazione del programma 19
- terminatore dell'istruzione (;) 47
- termostati smart 38
- testa di una coda (head) 523, 540
- testing 656
- testo
 - cifrato 382, 389
 - di sostituzione 216, 601
 - in chiaro 382, 388
- tgmath.h 636
- The Twelve Days of Christmas* 125
- Thompson, Ken 12
- thr_d_create, funzione 654
- thr_d_error 654
- thr_d_join, funzione 654
- thr_d_nomem 654
- thr_d_success 654
- thr_d_t, tipo 653
- thread XXXVII
 - di esecuzione 646
 - eseguire in parallelo XXXVII
 - ID 653
 - memoria locale 654
- _Thread_local, specificatore della classe di memoria 178
- <threads.h>, file di intestazione 647
- _TIME_, costante simbolica predefinita 607
- time, funzione del file di intestazione <time.h> 174
- <time.h>, file di intestazione 169
- Tiobe Index 1
- tipi 52
 - dei parametri 286
 - del valore di ritorno 161, 194
 - di dati aggregati 282, 423
 - di dati derivati 424, 433
 - di dati standard 288
 - di linguaggi di programmazione XXVI
 - in virgola mobile 635
 - interi portabili 636
- token 360, 363, 508 566, 607
 - compilatore Simple 566
- tokenizzazione 508
- tolower, funzione 345
- top 84
- top-down, affinamento graduale XXVII, 83, 85, 88, 89, 297, 299
- Torri di Hanoi 190, 205
- totale 83
- toupper, funzione 280, 345
- toziente 389
- traduzione 8 , 508
 - dei linguaggi 38
 - in tempo reale, 39

transazioni, sistema per l'elaborazione XXVIII, XXXVI
 trasferimento del controllo 73, 319, 320, 323
 trattare gli array di caratteri come stringhe 222
 trattino basso (_) 50
 triple pitagoriche 151
 troncare 87
 trovare il valore minimo in un array 269
true 636
 condizione 57
tvOS 11
typedef XXVIII 429

U

u o **U** per un **unsigned int** 621
Ubuntu Linux 23
 nel sottosistema Windows per Linux 23
#undef, direttiva per il preprocessore 603, 607
Unicode, 359
 insieme di caratteri 6, 124, 359
 supporto 635
unioni XXVIII, 433, 434, 455
unità
 aritmetica e logica (ALU) 4
 di input 3
 di memoria 425
 di output 4
 logiche 3
UNIX 125
 sistema operativo 12
UnloadGame, funzione in un gioco di raylib 462
unsigned int 164
unsigned long int 350
unsigned long long int 186, 187
unsigned short 164
UpdateGame, funzione in un gioco di raylib 462
URL (Uniform Resource Locator) 518
 usare il flag # 407
Utilities, area (Xcode) 24
 utilizzo della memoria 444

V

V dei big data 36
va_arg 614
`__VA_ARGS__` 640
va_copy, macro 643
va_end 614
va_list 614
va_start 614
valore 212
 assoluto 157

booleano JSON 518
chiave 237
 di flag 83
 di segnale 83
 di spostamento 174
 di una variabile 52, 53
finale 114
 finale di una variabile di controllo 117
fittizio 83
 iniziale di una variabile di controllo 114, 119
 maggiore, problema 68
 minimo in un array 190
 minore, problema 68
 previsto in regressione lineare semplice 514
 sentinella 83, 84, 86, 105
 "spazzatura" 83
 valutatore di espressioni postfisse, esercizio 561
valutazione
 cortocircuitata 132
 postfissa 567
 Van Rossum, Guido 15
variabili 50
 di istanza 657
 dipendenti 514
 globali 180, 181, 286, 617
 indipendenti 514
 inizializzazione delle 296
 locali 160, 161, 179, 180, 224
 nome 563, 565
 variabili di controllo 114, 118
 incremento 114
 nome 114
 valore iniziale 114
varianza 512
variazione di scala 172
varietà (nei big data) 36
vector2, tipo in raylib 461
 veicoli autonomi 4
 velocità (nei big data) 36
 veridicità (nei big data) 36
verità 131
 versione standard del C 13
Vi, editor 17
violazione
 alla restrizione 307
 di accesso 51, 342, 400, 401
virgola mobile 399
 limiti 169
 numeri 83, 86, 87, 88, 89
 specificatori di conversione 399, 403, 410
 suffisso f o F per un float 621
 suffisso l o L per un long double 621
Virtual Reality Modeling Language (VRML) 328

- visione artificiale 38, 42
- V**
- Visual C++
compilatore XXXIII, XLI, 20
linguaggio di programmazione 16
- Visual Studio 17
aggiungere un file esistente a un progetto 21
Community Edition XLI, XLV, 2, 20
compilare ed eseguire un programma 22
Empty Project, modello 20
filtrare modelli di progetti 20
finestra principale 21
progetti 20
prompt dei comandi, finestra 22
Search for templates 20
Solution Explorer 21
soluzioni 20
visualizzare Solution Explorer 21
- visualizzare
dati 38
ricorsione 190, 206
- visualizzazioni, XXXI, 38, 514
animate XXXV
con raylib: animazione della legge dei grandi numeri 465
- VLA (*variable-length array*), 249
- void * (puntatore a void) 291, 365, 525
- volume (nei big data) 36
- W**
- w, modalità di apertura dei file 480
w+, modalità di apertura dei file 480
- watchOS 11
- Watson di IBM XXXIII, 39, 42
- Waze, applicazione per la navigazione GPS 38
- wb, modalità di apertura dei file 480
wb+, modalità di apertura dei file 480
- web 517
- Webots XXXIV, 324
albero della scena 331, 332
campi 331
Create a Webots project directory, procedura
guidata 328
effetti di luce 339
e-puck_avoid_obstacles 334
evitare ostacoli 339
fisica 339
gravità 328
mondo 328
- motore fisico 334
nodi PROTO 332
nodo 331
opzioni fisiche 339
passo temporale di base 335
RectangleArena 327, 331
robot e-puck 327
robotica con simulatore XXXIV
ruote differenziali 327
texture 339
tour guidato 325
.wbt, file 328
WoodenBox 327, 332
- Welcome to Xcode, finestra 24
- while, istruzione di iterazione 80
- Windows, sistema operativo 10, 613, 621
- WindowShouldClose, funzione (raylib) 462
- WoodenBox (Webots) 327, 332
- World Wide Web 30
- World Wide Web Consortium (W3C) 30
- Wozniak, Steve 11
- write* 566
- X**
- Xcode XLI, XLV, 2, 17, 20
Command Line Tool, progetto 24
compilare ed eseguire un programma 25
Debug, area 25
Editor, area 24
finestra dello spazio di lavoro 24
Issue, navigatore 24
Navigator, area 24
progetto 24
Project, navigatore 24
Utilities, area 24
Welcome to Xcode, finestra 24
- Xerox PARC (*Palo Alto Research Center*) 11
- XOR bit a bit 436
- Y**
- y*, intercetta 514
- Z**
- 0 (zero) flag 407
zeri finali, 399
zettabyte (ZB) 33