

# Appunti di Programmazione C

Materiale Didattico per il Terzo Anno

Istituto Tecnico  
Anno Scolastico 2025-2026

8 novembre 2025

# Indice

<b>Prefazione</b>	<b>19</b>
<b>1 Introduzione al C</b>	<b>25</b>
1.1 Storia del Linguaggio C . . . . .	25
1.1.1 Timeline Storica . . . . .	25
1.2 Caratteristiche del Linguaggio C . . . . .	25
1.2.1 Punti di Forza . . . . .	26
1.2.2 Limitazioni . . . . .	26
1.3 Compilazione vs Interpretazione . . . . .	26
1.3.1 Linguaggi Interpretati . . . . .	26
1.3.2 Linguaggi Compilati . . . . .	27
1.3.3 Il Processo di Compilazione in C . . . . .	27
1.4 Ambiente di Sviluppo . . . . .	27
1.4.1 Installazione del Compilatore GCC . . . . .	28
1.4.2 Editor e IDE Consigliati . . . . .	28
1.5 Struttura di un Programma C . . . . .	29
1.5.1 Direttiva #include . . . . .	29
1.5.2 Funzione main() . . . . .	29
1.5.3 Commenti . . . . .	30
1.6 Il Primo Programma: Hello World . . . . .	30
1.6.1 Analisi del Codice . . . . .	30
1.6.2 Compilazione ed Esecuzione . . . . .	30
1.7 Esempi Guidati . . . . .	31
1.7.1 Esempio 1: Stampare Più Righe . . . . .	31
1.7.2 Esempio 2: Caratteri Speciali . . . . .	31
1.7.3 Esempio 3: Stampare Numeri . . . . .	32
1.8 Errori Comuni e Debugging . . . . .	32
1.8.1 Errore 1: Punto e Virgola Mancante . . . . .	32

1.8.2	Errore 2: Include Mancante . . . . .	33
1.8.3	Errore 3: Parentesi Graffe . . . . .	33
1.9	Librerie Standard del C . . . . .	33
1.10	Best Practices . . . . .	33
1.11	Esercizi . . . . .	34
1.11.1	Livello Base . . . . .	34
1.11.2	Livello Intermedio . . . . .	34
1.11.3	Livello Avanzato . . . . .	34
1.12	Riepilogo . . . . .	34
1.13	Approfondimenti . . . . .	35
<b>2</b>	<b>Variabili e Tipi di Dati</b>	<b>36</b>
2.1	Introduzione . . . . .	36
2.2	Obiettivi di Apprendimento . . . . .	36
2.3	Concetto di Variabile . . . . .	36
2.4	Tipi di Dati Base . . . . .	37
2.4.1	Tipi Interi . . . . .	37
2.4.2	Tipi a Virgola Mobile . . . . .	37
2.4.3	Tipo Carattere . . . . .	37
2.5	Dichiarazione di Variabili . . . . .	38
2.5.1	Sintassi Base . . . . .	38
2.5.2	Esempi di Dichiarazione . . . . .	38
2.5.3	Inizializzazione . . . . .	38
2.6	Regole per i Nomi delle Variabili . . . . .	38
2.6.1	Regole Obbligatorie . . . . .	39
2.6.2	Convenzioni Consigliate . . . . .	39
2.7	L'operatore sizeof . . . . .	39
2.8	Specificatori di Formato per printf . . . . .	40
2.8.1	Formattazione Avanzata . . . . .	40
2.9	Input da Tastiera: scanf . . . . .	41
2.9.1	Sintassi Base . . . . .	41
2.9.2	Esempi di Input . . . . .	41
2.10	Costanti . . . . .	42
2.10.1	Costanti con #define . . . . .	42
2.10.2	Costanti con const . . . . .	42
2.10.3	Differenza tra #define e const . . . . .	43

2.11 Conversioni di Tipo . . . . .	43
2.11.1 Conversione Implicita . . . . .	43
2.11.2 Conversione Esplicita (Cast) . . . . .	44
2.12 Esempi Pratici Completi . . . . .	44
2.12.1 Esempio 1: Calcolo Area Rettangolo . . . . .	44
2.12.2 Esempio 2: Conversione Temperature . . . . .	45
2.12.3 Esempio 3: Calcolo Media . . . . .	45
2.13 Overflow e Underflow . . . . .	46
2.13.1 Overflow . . . . .	46
2.13.2 Underflow . . . . .	46
2.14 Libreria limits.h . . . . .	46
2.15 Esercizi . . . . .	47
2.15.1 Livello Base . . . . .	47
2.15.2 Livello Intermedio . . . . .	47
2.15.3 Livello Avanzato . . . . .	47
2.16 Riepilogo . . . . .	48
2.17 Approfondimenti . . . . .	48
<b>3 Operatori ed Espressioni</b>	<b>49</b>
3.1 Introduzione agli Operatori . . . . .	49
3.2 Operatori Aritmetici . . . . .	49
3.2.1 Operatori Binari . . . . .	49
3.2.2 Operatori Unari . . . . .	49
3.2.3 Differenza tra Pre e Post Incremento . . . . .	50
3.3 Operatori di Assegnamento . . . . .	50
3.3.1 Assegnamento Semplice . . . . .	50
3.3.2 Operatori di Assegnamento Composto . . . . .	50
3.4 Operatori Relazionali . . . . .	50
3.5 Operatori Logici . . . . .	51
3.5.1 Valutazione in Cortocircuito . . . . .	51
3.6 Operatori Bit a Bit . . . . .	51
3.7 Operatore Ternario . . . . .	52
3.7.1 Sintassi . . . . .	52
3.7.2 Esempi . . . . .	52
3.8 Precedenza degli Operatori . . . . .	52
3.8.1 Tabella di Precedenza (dal più alto al più basso) . . . . .	52

---

3.8.2	Esempi di Precedenza . . . . .	53
3.9	Conversioni di Tipo . . . . .	53
3.9.1	Conversione Implicita (Casting Automatico) . . . . .	53
3.9.2	Conversione Esplicita (Cast) . . . . .	54
3.10	Espressioni Complesse . . . . .	54
3.10.1	Combinazione di Operatori . . . . .	54
3.10.2	Uso dell'Operatore sizeof . . . . .	54
3.11	Errori Comuni . . . . .	55
3.11.1	Confusione tra = e == . . . . .	55
3.11.2	Divisione Intera Inaspettata . . . . .	55
3.11.3	Overflow in Operazioni Aritmetiche . . . . .	55
3.12	Esercizi . . . . .	56
3.12.1	Livello Base . . . . .	56
3.12.2	Livello Intermedio . . . . .	56
3.12.3	Livello Avanzato . . . . .	56
<b>4</b>	<b>Controllo di Flusso</b> . . . . .	<b>57</b>
4.1	Introduzione . . . . .	57
4.1.1	Obiettivi di Apprendimento . . . . .	57
4.2	Istruzioni Condizionali . . . . .	57
4.2.1	Istruzione if . . . . .	57
4.2.2	Istruzione if-else . . . . .	58
4.2.3	Istruzione if-else if-else . . . . .	59
4.2.4	if Annidati . . . . .	60
4.2.5	Istruzione switch . . . . .	60
4.3	Cicli (Loop) . . . . .	62
4.3.1	Ciclo while . . . . .	62
4.3.2	Ciclo do-while . . . . .	63
4.3.3	Ciclo for . . . . .	64
4.4	Istruzioni di Controllo dei Cicli . . . . .	65
4.4.1	Istruzione break . . . . .	65
4.4.2	Istruzione continue . . . . .	66
4.4.3	Istruzione goto . . . . .	67
4.5	Confronto tra i Cicli . . . . .	67
4.5.1	Quando usare quale ciclo . . . . .	67
4.5.2	Equivalenza tra i Cicli . . . . .	68

4.6	Esempi Completi . . . . .	68
4.6.1	Calcolatrice con Menu . . . . .	68
4.6.2	Gioco: Indovina il Numero . . . . .	69
4.7	Esercizi . . . . .	70
4.7.1	Livello Base . . . . .	70
4.7.2	Livello Intermedio . . . . .	70
4.7.3	Livello Avanzato . . . . .	70
<b>5</b>	<b>Funzioni</b>	<b>71</b>
5.1	Introduzione alle Funzioni . . . . .	71
5.1.1	Obiettivi di apprendimento . . . . .	71
5.2	Definizione di una Funzione . . . . .	72
5.2.1	Sintassi . . . . .	72
5.2.2	Componenti di una Funzione . . . . .	72
5.2.3	Esempio Semplice . . . . .	72
5.3	Parametri e Argomenti . . . . .	72
5.3.1	Funzioni con Parametri . . . . .	72
5.3.2	Passaggio per Valore . . . . .	73
5.4	Valore di Ritorno . . . . .	73
5.4.1	Funzioni che Restituiscono un Valore . . . . .	73
5.4.2	Funzioni void . . . . .	74
5.5	Dichiarazione e Prototipo . . . . .	75
5.5.1	Prototipo di Funzione . . . . .	75
5.6	Funzioni Ricorsive . . . . .	75
5.6.1	Fattoriale Ricorsivo . . . . .	75
5.6.2	Fibonacci Ricorsivo . . . . .	76
5.7	Ambito delle Variabili (Scope) . . . . .	77
5.7.1	Variabili Locali . . . . .	77
5.7.2	Variabili Globali . . . . .	77
5.7.3	Variabili Static . . . . .	78
5.8	Esempi Pratici . . . . .	78
5.8.1	Calcolatrice con Funzioni . . . . .	78
5.8.2	Funzioni Matematiche . . . . .	80
5.9	Buone Pratiche . . . . .	81
5.9.1	Nomi Descrittivi . . . . .	81
5.9.2	Funzioni Piccole e Focalizzate . . . . .	81

5.9.3	Documentazione	81
5.10	Esercizi	82
5.10.1	Livello Base	82
5.10.2	Livello Intermedio	82
5.10.3	Livello Avanzato	82
<b>6</b>	<b>Array</b>	<b>83</b>
6.1	Introduzione agli Array	83
6.1.1	Obiettivi di Apprendimento	83
6.1.2	Caratteristiche degli Array	83
6.2	Dichiarazione e Inizializzazione	84
6.2.1	Sintassi Base	84
6.2.2	Esempi di Dichiarazione	84
6.3	Accesso agli Elementi	84
6.3.1	Lettura e Scrittura	84
6.3.2	Esempio di Errore	85
6.4	Attraversamento di un Array	85
6.4.1	Uso del Ciclo for	85
6.4.2	Dimensione Dinamica	85
6.5	Input e Output di Array	86
6.5.1	Lettura da Tastiera	86
6.6	Operazioni sugli Array	87
6.6.1	Ricerca del Massimo e del Minimo	87
6.6.2	Ricerca di un Elemento	87
6.6.3	Inversione di un Array	88
6.6.4	Copia di un Array	88
6.7	Array e Funzioni	89
6.7.1	Passaggio di Array a Funzioni	89
6.8	Array Multidimensionali	90
6.8.1	Array Bidimensionali (Matrici)	90
6.8.2	Operazioni su Matrici	90
6.9	Algoritmi di Ordinamento	92
6.9.1	Bubble Sort	92
6.9.2	Selection Sort	93
6.10	Esempi Pratici	93
6.10.1	Statistiche su un Array	93

6.10.2	Rimozione Duplicati	94
6.11	Esercizi	95
6.11.1	Livello Base	95
6.11.2	Livello Intermedio	95
6.11.3	Livello Avanzato	96
<b>7</b>	<b>Puntatori</b>	<b>97</b>
7.1	Introduzione ai Puntatori	97
7.1.1	Obiettivi di Apprendimento	97
7.1.2	Perché Usare i Puntatori?	97
7.2	Concetti Fondamentali	98
7.2.1	Indirizzi di Memoria	98
7.2.2	Operatori dei Puntatori	98
7.3	Dichiarazione e Inizializzazione	98
7.3.1	Sintassi	98
7.3.2	Esempi	98
7.3.3	Inizializzazione Diretta	99
7.4	Dereferenziazione	99
7.5	Puntatori e Funzioni	99
7.5.1	Passaggio per Riferimento	99
7.5.2	Scambio di Valori	100
7.5.3	Ritorno di Valori Multipli	101
7.6	Puntatori e Array	101
7.6.1	Relazione tra Array e Puntatori	101
7.6.2	Aritmetica dei Puntatori	102
7.6.3	Passaggio di Array a Funzioni con Puntatori	102
7.7	Puntatori a Puntatori	103
7.8	Puntatori void	103
7.9	Puntatori Costanti	104
7.9.1	Tipi di Costanti	104
7.10	Puntatore NULL	105
7.11	Allocazione Dinamica della Memoria	105
7.11.1	malloc	105
7.11.2	calloc	106
7.11.3	realloc	106
7.12	Errori Comuni con i Puntatori	107

7.12.1	Puntatore Non Inizializzato . . . . .	107
7.12.2	Dereferenziazione di NULL . . . . .	107
7.12.3	Memory Leak . . . . .	107
7.12.4	Dangling Pointer . . . . .	108
7.13	Esempi Pratici . . . . .	108
7.13.1	Lista Dinamica di Numeri . . . . .	108
7.14	Esercizi . . . . .	109
7.14.1	Livello Base . . . . .	109
7.14.2	Livello Intermedio . . . . .	109
7.14.3	Livello Avanzato . . . . .	109
<b>8</b>	<b>Stringhe</b>	<b>110</b>
8.1	Introduzione alle Stringhe . . . . .	110
8.1.1	Obiettivi di Apprendimento . . . . .	110
8.1.2	Caratteristiche delle Stringhe . . . . .	110
8.2	Dichiarazione e Inizializzazione . . . . .	111
8.2.1	Modi di Dichiarare Stringhe . . . . .	111
8.2.2	Differenza tra Array e Puntatore . . . . .	111
8.3	Input e Output di Stringhe . . . . .	112
8.3.1	Funzioni di I/O . . . . .	112
8.3.2	Carattere per Carattere . . . . .	112
8.4	Funzioni Standard per Stringhe . . . . .	113
8.4.1	strlen - Lunghezza della Stringa . . . . .	113
8.4.2	strcpy - Copia di Stringhe . . . . .	113
8.4.3	strcat - Concatenazione . . . . .	113
8.4.4	strcmp - Confronto di Stringhe . . . . .	114
8.4.5	Altre Funzioni Utili . . . . .	114
8.5	Manipolazione di Stringhe . . . . .	115
8.5.1	Conversione Maiuscolo/Minuscolo . . . . .	115
8.5.2	Inversione di una Stringa . . . . .	116
8.5.3	Rimozione degli Spazi . . . . .	116
8.5.4	Conta Parole . . . . .	117
8.6	Array di Stringhe . . . . .	117
8.6.1	Array Bidimensionale di char . . . . .	117
8.6.2	Array di Puntatori a Stringhe . . . . .	118
8.7	Conversioni tra Stringhe e Numeri . . . . .	118

8.7.1	Da Stringa a Numero . . . . .	118
8.7.2	Da Numero a Stringa . . . . .	119
8.8	Tokenizzazione di Stringhe . . . . .	119
8.8.1	Uso di strtok . . . . .	119
8.9	Validazione di Stringhe . . . . .	120
8.9.1	Verifica se è un Numero . . . . .	120
8.9.2	Verifica Palindromo . . . . .	121
8.10	Esempi Pratici . . . . .	121
8.10.1	Sistema di Login . . . . .	121
8.10.2	Analisi di Testo . . . . .	122
8.11	Esercizi . . . . .	123
8.11.1	Livello Base . . . . .	123
8.11.2	Livello Intermedio . . . . .	123
8.11.3	Livello Avanzato . . . . .	123
<b>9</b>	<b>Struct</b> . . . . .	<b>124</b>
9.1	Obiettivi di Apprendimento . . . . .	124
9.2	Introduzione alle Struct . . . . .	124
9.2.1	Perché Usare le Struct? . . . . .	124
9.3	Definizione di una Struct . . . . .	125
9.3.1	Sintassi Base . . . . .	125
9.3.2	Esempi di Definizione . . . . .	125
9.4	Dichiarazione e Inizializzazione . . . . .	126
9.4.1	Dichiarazione . . . . .	126
9.4.2	Inizializzazione . . . . .	126
9.5	Accesso ai Membri . . . . .	127
9.5.1	Operatore Punto (.) . . . . .	127
9.5.2	Input da Utente . . . . .	127
9.6	Typedef . . . . .	128
9.7	Array di Struct . . . . .	129
9.8	Struct Annidate . . . . .	130
9.9	Puntatori a Struct . . . . .	131
9.9.1	Operatore Freccia (->) . . . . .	131
9.10	Struct e Funzioni . . . . .	132
9.10.1	Passaggio per Valore . . . . .	132
9.10.2	Passaggio per Riferimento . . . . .	133

9.10.3 Ritorno di Struct . . . . .	133
9.11 Allocazione Dinamica di Struct . . . . .	134
9.12 Esempi Pratici . . . . .	136
9.12.1 Sistema di Gestione Studenti . . . . .	136
9.12.2 Sistema di Coordinate 3D . . . . .	137
9.13 Esercizi . . . . .	138
9.13.1 Livello Base . . . . .	138
9.13.2 Livello Intermedio . . . . .	139
9.13.3 Livello Avanzato . . . . .	139
<b>10 Gestione dei File</b>	<b>140</b>
10.1 Introduzione ai File . . . . .	140
10.1.1 Tipi di File . . . . .	140
10.2 Apertura e Chiusura di File . . . . .	140
10.2.1 Funzione fopen . . . . .	140
10.2.2 Modalità di Apertura . . . . .	140
10.2.3 Esempio Base . . . . .	141
10.3 Scrittura su File . . . . .	141
10.3.1 fprintf . . . . .	141
10.3.2 fputs . . . . .	142
10.3.3 fputc . . . . .	142
10.4 Lettura da File . . . . .	143
10.4.1 fscanf . . . . .	143
10.4.2 fgets . . . . .	143
10.4.3 fgetc . . . . .	144
10.5 Controllo Fine File e Errori . . . . .	144
10.5.1 feof e ferror . . . . .	144
10.6 Posizionamento nel File . . . . .	145
10.6.1 fseek . . . . .	145
10.6.2 ftell . . . . .	145
10.7 File Binari . . . . .	146
10.7.1 fwrite . . . . .	146
10.7.2 fread . . . . .	147
10.8 Esempi Pratici . . . . .	147
10.8.1 Copia di File . . . . .	147
10.8.2 Conta Righe, Parole e Caratteri . . . . .	148

10.8.3 Database Studenti su File . . . . .	149
10.8.4 File CSV . . . . .	151
10.9 Gestione Errori . . . . .	152
10.9.1 perror . . . . .	152
10.10Esercizi . . . . .	152
10.10.1 Livello Base . . . . .	152
10.10.2 Livello Intermedio . . . . .	152
10.10.3 Livello Avanzato . . . . .	153
<b>11 Esercizi Completi</b>	<b>154</b>
11.1 Variabili e Tipi di Dati . . . . .	154
11.1.1 Livello Base . . . . .	154
11.1.2 Livello Intermedio . . . . .	154
11.1.3 Livello Avanzato . . . . .	154
11.2 Operatori ed Espressioni . . . . .	155
11.2.1 Livello Base . . . . .	155
11.2.2 Livello Intermedio . . . . .	155
11.2.3 Livello Avanzato . . . . .	155
11.3 Controllo di Flusso . . . . .	155
11.3.1 Livello Base . . . . .	155
11.3.2 Livello Intermedio . . . . .	156
11.3.3 Livello Avanzato . . . . .	156
11.4 Funzioni . . . . .	156
11.4.1 Livello Base . . . . .	156
11.4.2 Livello Intermedio . . . . .	156
11.4.3 Livello Avanzato . . . . .	156
11.5 Array . . . . .	157
11.5.1 Livello Base . . . . .	157
11.5.2 Livello Intermedio . . . . .	157
11.5.3 Livello Avanzato . . . . .	157
11.6 Puntatori . . . . .	157
11.6.1 Livello Base . . . . .	157
11.6.2 Livello Intermedio . . . . .	157
11.6.3 Livello Avanzato . . . . .	158
11.7 Stringhe . . . . .	158
11.7.1 Livello Base . . . . .	158

11.7.2 Livello Intermedio . . . . .	158
11.7.3 Livello Avanzato . . . . .	158
11.8 Struct . . . . .	158
11.8.1 Livello Base . . . . .	158
11.8.2 Livello Intermedio . . . . .	159
11.8.3 Livello Avanzato . . . . .	159
11.9 File . . . . .	159
11.9.1 Livello Base . . . . .	159
11.9.2 Livello Intermedio . . . . .	159
11.9.3 Livello Avanzato . . . . .	159
11.10 Progetti Completi . . . . .	160
11.10.1 Progetto 1: Sistema di Gestione Biblioteca . . . . .	160
11.10.2 Progetto 2: Gioco del Tris . . . . .	160
11.10.3 Progetto 3: Gestionale Studenti . . . . .	160
11.10.4 Progetto 4: Calcolatrice Avanzata . . . . .	161
11.10.5 Progetto 5: Sistema di Prenotazioni . . . . .	161
11.11 Esercizi di Riepilogo . . . . .	161
11.11.1 Esercizio Complessivo 1: Gestione Magazzino . . . . .	161
11.11.2 Esercizio Complessivo 2: Agenda Personale . . . . .	162
11.11.3 Esercizio Complessivo 3: Analizzatore di Testo . . . . .	162
11.12 Consigli per gli Esercizi . . . . .	162
11.12.1 Come Affrontare gli Esercizi . . . . .	162
11.12.2 Debugging . . . . .	162
11.12.3 Best Practices . . . . .	163
11.13 Risorse Aggiuntive . . . . .	163
11.13.1 Compilazione . . . . .	163
11.13.2 Online Judge per Esercitarsi . . . . .	164
11.13.3 Documentazione di Riferimento . . . . .	164
<b>12 Makefile e Build Automation</b>	<b>165</b>
12.1 Introduzione ai Makefile . . . . .	165
12.1.1 Obiettivi di apprendimento . . . . .	165
12.2 Struttura Base di un Makefile . . . . .	166
12.2.1 Sintassi di una Regola . . . . .	166
12.2.2 Esempio Semplice . . . . .	166
12.2.3 Esecuzione . . . . .	166

12.3 Variabili e Pattern . . . . .	166
12.3.1 Variabili Comuni . . . . .	166
12.3.2 Pattern Rules . . . . .	167
12.3.3 Variabili Integrate . . . . .	167
12.4 Makefile Completo con Esercizi . . . . .	168
12.4.1 Progetto Multi-File . . . . .	168
12.4.2 Progetto con Sottodirectory . . . . .	169
12.4.3 Makefile Avanzato con Documentazione . . . . .	170
12.5 Dipendenze e Ottimizzazione . . . . .	171
12.5.1 File di Dipendenza Automatiche . . . . .	171
12.5.2 Compilazione Parallela . . . . .	172
12.6 Esercizi . . . . .	172
12.6.1 Livello Base . . . . .	172
12.6.2 Livello Intermedio . . . . .	172
12.6.3 Livello Avanzato . . . . .	173
12.7 Riepilogo . . . . .	173
<b>13 GDB Debugger</b>	<b>174</b>
13.1 Introduzione al Debugging . . . . .	174
13.1.1 Obiettivi di apprendimento . . . . .	174
13.2 Compilazione per il Debugging . . . . .	175
13.2.1 Flag -g . . . . .	175
13.2.2 Esempio Programma con Bug . . . . .	175
13.3 Comandi Fondamentali di GDB . . . . .	176
13.3.1 Avviare GDB . . . . .	176
13.3.2 Breakpoint . . . . .	176
13.3.3 Esecuzione del Programma . . . . .	176
13.3.4 Ispezionare Variabili . . . . .	177
13.3.5 Watchpoint . . . . .	177
13.4 Stack e Funzioni . . . . .	177
13.4.1 Stack Trace . . . . .	177
13.4.2 Esempio di Debugging con Stack . . . . .	178
13.5 Comandi Avanzati . . . . .	178
13.5.1 Memoria e Puntatori . . . . .	178
13.5.2 Modificare Valori . . . . .	179
13.5.3 Script e Automazione . . . . .	179

13.6 Esempi Pratici di Debugging . . . . .	179
13.6.1 Debug di Segmentation Fault . . . . .	179
13.6.2 Debug di Buffer Overflow . . . . .	180
13.6.3 Debug di Loop Infinito . . . . .	180
13.7 Comandi Utili Riassunti . . . . .	181
13.7.1 Tabella Veloce . . . . .	181
13.8 Esercizi . . . . .	181
13.8.1 Livello Base . . . . .	181
13.8.2 Livello Intermedio . . . . .	182
13.8.3 Livello Avanzato . . . . .	182
13.9 Riepilogo . . . . .	182
<b>14 Librerie Standard Avanzate . . . . .</b>	<b>183</b>
14.1 Introduzione alle Librerie Standard . . . . .	183
14.1.1 Obiettivi di apprendimento . . . . .	183
14.2 string.h - Manipolazione di Stringhe . . . . .	184
14.2.1 Funzioni Base . . . . .	184
14.2.2 Copia di Stringhe . . . . .	184
14.2.3 Concatenazione e Ricerca . . . . .	184
14.2.4 Funzioni Avanzate . . . . .	185
14.3 math.h - Operazioni Matematiche . . . . .	186
14.3.1 Funzioni Trigonometriche . . . . .	186
14.3.2 Potenze e Radici . . . . .	186
14.3.3 Logaritmi e Funzioni Esponenziali . . . . .	187
14.3.4 Funzioni di Arrotondamento e Valore Assoluto . . . . .	187
14.4 time.h - Gestione del Tempo . . . . .	188
14.4.1 Funzioni Base di Tempo . . . . .	188
14.4.2 Misurazione del Tempo Trascorso . . . . .	188
14.4.3 Formattazione della Data . . . . .	189
14.5 stdlib.h - Allocazione Memoria e Utilità . . . . .	189
14.5.1 Allocazione Dinamica . . . . .	189
14.5.2 calloc e realloc . . . . .	190
14.5.3 Ordinamento e Ricerca . . . . .	191
14.5.4 bsearch - Ricerca Binaria . . . . .	191
14.6 Esempio Completo: Gestione Database di Studenti . . . . .	192
14.7 Esercizi . . . . .	194

14.7.1 Livello Base . . . . .	194
14.7.2 Livello Intermedio . . . . .	194
14.7.3 Livello Avanzato . . . . .	194
14.8 Riepilogo . . . . .	194
<b>Bibliografia e Risorse</b>	<b>196</b>
<b>Appendice: Soluzioni ad Alcuni Esercizi</b>	<b>202</b>

# Elenco delle figure

1.1 Fasi del processo di compilazione . . . . .	27
---	----

# Listings

1	Esempio di codice C . . . . .	23
1.1	Installazione GCC su Ubuntu/Debian . . . . .	28
1.2	Installazione GCC su macOS . . . . .	28
1.3	Struttura base di un programma C . . . . .	29
1.4	Tipi di commenti in C . . . . .	30
1.5	Hello World in C . . . . .	30
1.6	Compilazione ed esecuzione . . . . .	31
1.7	Programma con più stampe . . . . .	31
1.8	Uso di caratteri speciali . . . . .	31
1.9	Stampa di numeri . . . . .	32
1.10	Errore: punto e virgola mancante . . . . .	32
1.11	Errore: header mancante . . . . .	33
1.12	Errore: parentesi graffe non bilanciate . . . . .	33
2.1	Esempio concettuale di variabile . . . . .	36
2.2	Tipo char . . . . .	37
2.3	Sintassi di dichiarazione . . . . .	38
2.4	Dichiarazione di variabili . . . . .	38
2.5	Inizializzazione di variabili . . . . .	38
2.6	Esempi di nomi validi e non . . . . .	39
2.7	Uso di sizeof . . . . .	39
2.8	Controllo del formato di stampa . . . . .	40
2.9	Sintassi di scanf . . . . .	41
2.10	Leggere input con scanf . . . . .	41
2.11	Costanti con #define . . . . .	42
2.12	Costanti con const . . . . .	42
2.13	Conversione implicita . . . . .	43
2.14	Cast esplicito . . . . .	44
2.15	Calcolo area e perimetro di un rettangolo . . . . .	44

---

2.16 Conversione da Celsius a Fahrenheit . . . . .	45
2.17 Calcolo media di tre numeri . . . . .	45
2.18 Esempio di overflow . . . . .	46
2.19 Esempio di underflow . . . . .	46
2.20 Uso di limits.h . . . . .	46

# Prefazione

## Benvenuti al Corso di Programmazione C

Questo libro nasce dall'esigenza di fornire agli studenti del terzo anno dell'Istituto Tecnico un materiale didattico completo, chiaro e pratico per l'apprendimento del linguaggio C. Il C è un linguaggio di programmazione fondamentale, la cui conoscenza rappresenta una solida base per qualsiasi percorso nel mondo della programmazione e dell'informatica.

## Perché Studiare il C?

Il linguaggio C, nato nei primi anni '70 nei laboratori Bell, è ancora oggi uno dei linguaggi più utilizzati e influenti nella storia dell'informatica. Nonostante la sua età, il C rimane attuale per diversi motivi:

- **Fondamenta solide:** Il C insegna i concetti fondamentali della programmazione procedurale e della gestione della memoria.
- **Efficienza:** È uno dei linguaggi più veloci ed efficienti, utilizzato per sistemi operativi, driver e software embedded.
- **Portabilità:** Il codice C può essere compilato su praticamente qualsiasi piattaforma.
- **Controllo:** Offre un controllo diretto sull'hardware e sulla memoria.
- **Base per altri linguaggi:** C++, Java, C# e molti altri linguaggi derivano dal C o ne condividono la sintassi.
- **Richiesto nel mondo professionale:** Molte aziende cercano programmatori C, specialmente nei settori automotive, aerospaziale, telecomunicazioni e sistemi embedded.

### Nota

Imparare il C vi renderà programmatori migliori, indipendentemente dal linguaggio che userete in futuro. Il C vi obbliga a comprendere realmente cosa succede "sotto il cofano" del computer.

## A Chi si Rivolge Questo Libro

Questo materiale è stato pensato specificamente per studenti del terzo anno di un Istituto Tecnico con indirizzo informatico. I prerequisiti richiesti sono:

- Conoscenza di base della matematica (aritmetica, algebra elementare)
- Capacità di ragionamento logico
- Familiarità con l'uso del computer
- Voglia di imparare e sperimentare

**Non è richiesta** alcuna esperienza precedente di programmazione. Partiremo dalle basi e procederemo gradualmente verso argomenti più complessi.

## Struttura del Libro

Il materiale è organizzato in 11 capitoli, progettati per essere affrontati sequenzialmente:

**Capitolo 1 - Introduzione al C** Storia del linguaggio, ambiente di sviluppo, primo programma.

**Capitolo 2 - Variabili e Tipi di Dati** Dichiarazione variabili, tipi base, input/output.

**Capitolo 3 - Operatori ed Espressioni** Operatori aritmetici, logici, relazionali e precedenza.

**Capitolo 4 - Controllo di Flusso** Strutture condizionali (if, switch) e cicli (for, while).

**Capitolo 5 - Array** Array monodimensionali e multidimensionali.

**Capitolo 6 - Funzioni** Dichiarazione, definizione, parametri, ricorsione.

**Capitolo 7 - Puntatori** Concetti fondamentali, dereferenziazione, aritmetica dei puntatori.

**Capitolo 8 - Stringhe** Manipolazione stringhe, funzioni della libreria standard.

**Capitolo 9 - Struct** Strutture dati personalizzate, typedef, union, enum.

**Capitolo 10 - File** Input/output su file, gestione degli errori.

**Capitolo 11 - Preprocessore** Direttive, macro, compilazione condizionale.

Ogni capitolo include:

- Spiegazioni teoriche chiare e concise
- Esempi di codice commentati
- Box informativi (attenzione, note, errori comuni)
- Esercizi di difficoltà crescente
- Riepilogo finale

## Come Usare Questo Libro

Per ottenere il massimo da questo materiale didattico, segui questi consigli:

## 1. Leggi Attivamente

Non limitarti a leggere passivamente. Prendi appunti, sottolinea, fai schemi. Cerca di riformulare i concetti con parole tue.

## 2. Scrivi il Codice

### Attenzione

NON copiare e incollare il codice dagli esempi! Digitalo personalmente, carattere per carattere. Questo ti aiuterà a:

- Memorizzare la sintassi
- Notare i dettagli
- Fare esperienza con gli errori di compilazione
- Sviluppare "memoria muscolare" per la programmazione

## 3. Sperimenta

Una volta che un esempio funziona:

- Modificalo
- Prova a "romperlo"
- Cambia i valori
- Aggiungi funzionalità
- Chiediti "cosa succede se..."

La sperimentazione è fondamentale per l'apprendimento della programmazione.

## 4. Fai Gli Esercizi

Gli esercizi sono organizzati in tre livelli:

**Base** Applicazione diretta dei concetti spiegati nel capitolo. Sono essenziali per verificare di aver compreso le basi.

**Intermedio** Richiedono di combinare più concetti o di ragionare un po' di più. Sono il cuore dell'apprendimento.

**Avanzato** Problemi più complessi che richiedono creatività e ragionamento. Sono sfide che vi prepareranno per situazioni reali.

**Strategia consigliata:**

1. Prova a risolvere l'esercizio autonomamente

2. Se ti blocchi, rileggi la sezione teorica
3. Se ancora non riesci, consulta la soluzione nell'appendice
4. Analizza la soluzione e cerca di capire ogni riga
5. Prova a riscrivere la soluzione senza guardarla

## 5. Gestisci gli Errori

### Nota

Gli errori sono **normali** e **utili**! Ogni programmatore, dal principiante al professionista, incontra continuamente errori. La differenza sta nel saperli affrontare:

- Leggi attentamente i messaggi di errore del compilatore
- Google è tuo amico: cerca l'errore online
- Controlla riga per riga il codice
- Usa printf per "debuggare" e capire cosa sta succedendo
- Chiedi aiuto ai compagni o all'insegnante

## 6. Pratica Costante

La programmazione è come imparare a suonare uno strumento: richiede pratica regolare. Dedica tempo ogni giorno, anche solo 30 minuti, piuttosto che sessioni lunghe saltuarie.

## Ambiente di Sviluppo

Per seguire questo corso avrai bisogno di:

- **Un compilatore C:** Consigliamo GCC (GNU Compiler Collection)
  - Linux: già installato o disponibile tramite package manager
  - macOS: installabile tramite Xcode Command Line Tools
  - Windows: MinGW o Cygwin
- **Un editor di testo o IDE:**
  - Visual Studio Code (consigliato, gratuito, multiplatforma)
  - Code::Blocks (IDE specifico per C/C++, gratuito)
  - Dev-C++ (solo Windows, semplice per iniziare)
  - CLion (professionale, a pagamento ma gratuito per studenti)

Nel Capitolo 1 troverai le istruzioni dettagliate per installare e configurare l'ambiente di sviluppo.

## Convenzioni Tipografiche

In questo libro utilizziamo diverse convenzioni per rendere il contenuto più chiaro:

- **Codice inline:** parole chiave, nomi di variabili, funzioni (es. `printf`, `int`, `main`)
- **Grassetto:** concetti importanti, termini tecnici alla prima occorrenza
- *Corsivo:* enfasi, termini stranieri

## Box Informativi

Troverai diversi tipi di box colorati:

### Attenzione

I box **Attenzione** (arancione) evidenziano concetti critici, errori da evitare o informazioni particolarmente importanti che non devono essere trascurate.

### Nota

I box **Nota** (blu) contengono suggerimenti, consigli pratici, best practices e informazioni utili che arricchiscono la comprensione.

### Errore Comune

I box **Errore Comune** (rosso) descrivono errori tipici che i principianti (e a volte anche i programmatori esperti!) commettono frequentemente, aiutandoti a evitarli.

## Esempi di Codice

Gli esempi di codice sono presentati in box con sfondo grigio chiaro, numerazione delle righe e sintassi colorata:

```
1 #include <stdio.h>
2
3 int main() {
4     // Questo e' un commento
5     printf("Ciao, mondo!\n"); // Stampa un messaggio
6     return 0;
7 }
```

Listing 1: Esempio di codice C

## Risorse Online

Oltre a questo libro, ti consigliamo di consultare:

- **C Reference:** <https://en.cppreference.com/w/c>
- **Learn C:** <https://www.learn-c.org/>

- **Stack Overflow:** <https://stackoverflow.com/questions/tagged/c>
- **Online GDB:** <https://www.onlinegdb.com/> (compilatore online)

## Un Ultimo Consiglio

*“La programmazione è un’arte pratica.  
Non imparerai a programmare leggendo,  
ma scrivendo codice.”*

Non scoraggiarti se all’inizio alcuni concetti sembrano difficili. Il C è un linguaggio che richiede tempo e pratica per essere padroneggiato. La frustrazione è parte del processo di apprendimento. Ogni errore risolto è una lezione appresa.

Prenditi il tempo necessario, procedi con calma, e soprattutto: **divertiti!** La programmazione è un’attività creativa e gratificante. Vedrai che, capitolo dopo capitolo, diventerai sempre più sicuro e capace.

Buono studio e buona programmazione!

*L’Autore*

*Novembre 2025*

# Capitolo 1

## Introduzione al C

### 1.1 Storia del Linguaggio C

Il linguaggio C è stato sviluppato tra il 1969 e il 1973 da **Dennis Ritchie** nei laboratori Bell della AT&T. Inizialmente creato per riscrivere il sistema operativo UNIX, il C è rapidamente diventato uno dei linguaggi più influenti nella storia dell'informatica.

#### 1.1.1 Timeline Storica

- **1969-1973:** Dennis Ritchie sviluppa il C basandosi sul linguaggio B (creato da Ken Thompson)
- **1973:** Il sistema operativo UNIX viene riscritto in C
- **1978:** Brian Kernighan e Dennis Ritchie pubblicano "The C Programming Language" (il famoso "K&R")
- **1989:** L'ANSI pubblica lo standard C89/C90 (ANSI C)
- **1999:** Pubblicazione dello standard C99
- **2011:** Pubblicazione dello standard C11
- **2018:** Pubblicazione dello standard C17/C18

#### Nota

Il C è considerato un linguaggio di "livello medio": offre il controllo dell'hardware tipico dei linguaggi di basso livello (come l'Assembly) mantenendo la leggibilità e la portabilità dei linguaggi di alto livello.

### 1.2 Caratteristiche del Linguaggio C

Il C presenta caratteristiche che lo hanno reso estremamente popolare e longevo:

### 1.2.1 Punti di Forza

**Efficienza** Il codice C è molto veloce ed efficiente. I programmi compilati in C sono tra i più performanti.

**Portabilità** Il codice C può essere compilato su praticamente qualsiasi piattaforma (Windows, Linux, macOS, microcontrollori, etc.).

**Controllo** Offre controllo diretto sulla memoria e sull'hardware.

**Semplicità** La sintassi è relativamente semplice con poche parole chiave (circa 32 nello standard C89).

**Libreria Standard** Fornisce una libreria standard ricca di funzioni utili.

**Base per altri linguaggi** C++, Objective-C, C#, Java e molti altri derivano dal C.

### 1.2.2 Limitazioni

**Gestione manuale della memoria** Il programmatore deve allocare e deallocare la memoria manualmente.

**Assenza di OOP nativa** Non supporta nativamente la programmazione orientata agli oggetti.

**Mancanza di controlli automatici** Non verifica automaticamente i limiti degli array.

**Sintassi a volte criptica** Alcune costruzioni possono essere difficili da leggere per i principianti.

#### Attenzione

Il C offre grande potere, ma con esso viene grande responsabilità. Errori nella gestione della memoria o nell'uso dei puntatori possono causare bug difficili da individuare.

## 1.3 Compilazione vs Interpretazione

Prima di procedere, è importante capire la differenza tra linguaggi compilati e interpretati.

### 1.3.1 Linguaggi Interpretati

- Il codice viene eseguito riga per riga da un **interprete**
- Esempi: Python, JavaScript, Ruby
- **Vantaggio:** Sviluppo rapido, facile debugging
- **Svantaggio:** Generalmente più lenti

### 1.3.2 Linguaggi Compilati

- Il codice viene tradotto tutto insieme in linguaggio macchina da un **compilatore**
- Il risultato è un file eseguibile
- Esempi: C, C++, Rust
- **Vantaggio:** Molto più veloci in esecuzione
- **Svantaggio:** Processo di compilazione necessario

### 1.3.3 Il Processo di Compilazione in C

La compilazione di un programma C avviene in 4 fasi:

1. **Preprocessore:** Elabora le direttive (linee che iniziano con #)
2. **Compilazione:** Traduce il codice C in Assembly
3. **Assemblaggio:** Converte l'Assembly in codice oggetto (file .o o .obj)
4. **Linking:** Collega il codice oggetto con le librerie creando l'eseguibile finale

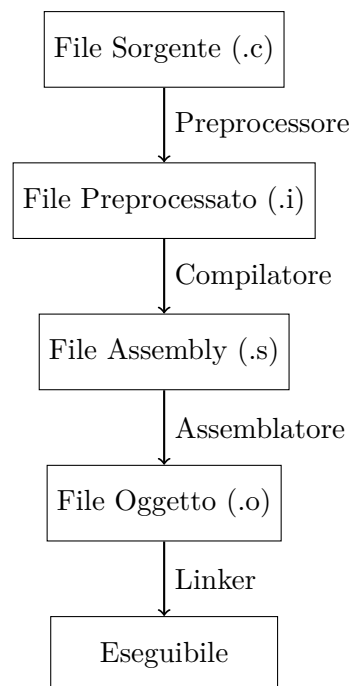


Figura 1.1: Fasi del processo di compilazione

## 1.4 Ambiente di Sviluppo

Per programmare in C hai bisogno di due strumenti fondamentali:

1. Un **editor di testo** o **IDE** (Integrated Development Environment)
2. Un **compilatore C**

### 1.4.1 Installazione del Compilatore GCC

GCC (GNU Compiler Collection) è il compilatore C più utilizzato, gratuito e open source.

#### Su Linux

```
1 # Aggiorna i repository
2 sudo apt update
3
4 # Installa build-essential (include GCC)
5 sudo apt install build-essential
6
7 # Verifica l'installazione
8 gcc --version
```

Listing 1.1: Installazione GCC su Ubuntu/Debian

#### Su macOS

```
1 # Installa Xcode Command Line Tools
2 xcode-select --install
3
4 # Verifica l'installazione
5 gcc --version
```

Listing 1.2: Installazione GCC su macOS

#### Su Windows

##### Opzione 1: MinGW

1. Scarica MinGW da <http://www.mingw.org/>
2. Installa e aggiungi al PATH
3. Verifica con `gcc --version`

##### Opzione 2: WSL (Windows Subsystem for Linux)

1. Abilita WSL nelle impostazioni Windows
2. Installa una distribuzione Linux (es. Ubuntu)
3. Segui le istruzioni per Linux

### 1.4.2 Editor e IDE Consigliati

**Visual Studio Code** Gratuito, leggero, multiplatforma. Installa l'estensione "C/C++" di Microsoft.

**Code::Blocks** IDE gratuito specifico per C/C++, semplice e completo.

**Dev-C++** IDE leggero per Windows, ideale per iniziare (ormai obsoleto ma ancora usato in didattica).

**CLion** IDE professionale di JetBrains, gratuito per studenti.

#### Nota

Per questo corso consigliamo **Visual Studio Code** per la sua versatilità, o **Code::Blocks** per la semplicità d'uso.

## 1.5 Struttura di un Programma C

Analizziamo la struttura base di un programma C:

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     // Codice del programma
5     return 0;
6 }
```

Listing 1.3: Struttura base di un programma C

Analizziamo ogni componente:

### 1.5.1 Direttiva #include

```
1 #include <stdio.h>
```

- `#include` è una direttiva del preprocessore
- Include il contenuto di un file header
- `<stdio.h>` è la libreria Standard Input/Output
- Contiene le dichiarazioni delle funzioni della libreria stdio come `printf()` e `scanf()`

### 1.5.2 Funzione main()

```
1 int main() {
2     // ...
3     return 0;
4 }
```

- `main()` è la funzione principale del programma
- L'esecuzione inizia sempre da `main()`
- `int` indica che la funzione restituisce un intero
- `return 0` indica che il programma è terminato correttamente

- Le parentesi graffe `{...}` delimitano il corpo della funzione

#### Attenzione

Ogni programma C deve avere una (e una sola) funzione `main()`. Senza di essa, il programma non può essere eseguito.

### 1.5.3 Commenti

Il C supporta due tipi di commenti:

```
1 // Commento su una singola riga (stile C99)
2
3 /* Commento su
4    piu' righe
5    (stile C89) */
6
7 int x = 5; // Commento a fine riga
```

Listing 1.4: Tipi di commenti in C

I commenti sono ignorati dal compilatore e servono per documentare il codice.

## 1.6 Il Primo Programma: Hello World

Scriviamo il tradizionale primo programma:

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     printf("Hello, World!\n");
5     return 0;
6 }
```

Listing 1.5: Hello World in C

### 1.6.1 Analisi del Codice

`printf()` La funzione `printf()` della libreria `stdio` stampa testo sullo schermo

`"Hello, World!"` La stringa da stampare (racchiusa tra virgolette)

`\n` Il carattere speciale: va a capo (newline)

`;` Il punto e virgola: termina ogni istruzione in C

### 1.6.2 Compilazione ed Esecuzione

Da Terminale

```
1 # Compila il programma
2 gcc hello.c -o hello
3
4 # Esegui il programma
5 ./hello           # Su Linux/macOS
6 hello.exe         # Su Windows
```

Listing 1.6: Compilazione ed esecuzione

Opzioni comuni di GCC:

- `-o nome`: specifica il nome del file eseguibile
- `-Wall`: abilita tutti i warning
- `-Wextra`: abilita warning extra
- `-g`: include informazioni di debug

#### Nota

È buona pratica compilare sempre con `-Wall -Wextra` per individuare potenziali errori:

```
1 gcc -Wall -Wextra hello.c -o hello
```

## 1.7 Esempi Guidati

### 1.7.1 Esempio 1: Stampare Più Righe

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     printf("Benvenuto nel mondo del C!\n");
5     printf("Questo e' il mio primo programma.\n");
6     printf("La programmazione e' divertente!\n");
7     return 0;
8 }
```

Listing 1.7: Programma con più stampe

Output:

```
Benvenuto nel mondo del C!
Questo e' il mio primo programma.
La programmazione e' divertente!
```

### 1.7.2 Esempio 2: Caratteri Speciali

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     printf("Riga 1\n");           // \n va a capo
```

```
5 printf("Riga 2\tcon tab\n"); // \t inserisce una tabulazione
6 printf("Virgolette: \"Ciao\"\n"); // \" stampa le virgolette
7 printf("Backslash: \\n"); // \\ stampa un backslash
8 return 0;
9 }
```

Listing 1.8: Uso di caratteri speciali

Output:

```
Riga 1
Riga 2    con tab
Virgolette: "Ciao"
Backslash: \
```

### 1.7.3 Esempio 3: Stampare Numeri

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     printf("Il numero e': %d\n", 42);
5     printf("Pi greco: %.2f\n", 3.14159);
6     printf("Carattere: %c\n", 'A');
7     return 0;
8 }
```

Listing 1.9: Stampa di numeri

Output:

```
Il numero e': 42
Pi greco: 3.14
Carattere: A
```

#### Nota

Gli specificatori di formato della funzione printf() (%d, %f, %c) verranno approfonditi nel [Capitolo 2](#).

## 1.8 Errori Comuni e Debugging

### 1.8.1 Errore 1: Punto e Virgola Mancante

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Ciao") // ERRORE: manca il ;
5     return 0;
6 }
```

Listing 1.10: Errore: punto e virgola mancante

Messaggio di errore tipico del compilatore:

```
error: expected ';' before 'return'
```

### 1.8.2 Errore 2: Include Mancante

```

1 // ERRORE: manca #include <stdio.h>
2
3 int main() {
4     printf("Ciao\n"); // La funzione printf() non e' dichiarata!
5     return 0;
6 }
```

Listing 1.11: Errore: header mancante

### 1.8.3 Errore 3: Parentesi Graffe

```

1 #include <stdio.h>
2
3 int main() {
4     printf("Ciao\n");
5     return 0;
6 // ERRORE: manca la } di chiusura
```

Listing 1.12: Errore: parentesi graffe non bilanciate

#### Errore Comune

Gli errori di sintassi più comuni per i principianti sono:

- Dimenticare il punto e virgola
- Non bilanciare le parentesi (tonde, quadre, graffe)
- Dimenticare il `#include <stdio.h>`
- Usare virgolette semplici per stringhe (usare doppie!)

## 1.9 Librerie Standard del C

Il C fornisce diverse librerie standard utili:

Libreria	Funzionalità
<code>stdio.h</code>	Input/Output (funzioni <code>printf()</code> , <code>scanf()</code> , <code>fopen()</code> , etc.)
<code>stdlib.h</code>	Utility generali (funzioni <code>malloc()</code> , <code>free()</code> , <code>rand()</code> , etc.)
<code>string.h</code>	Manipolazione stringhe (funzioni <code>strlen()</code> , <code>strcpy()</code> , etc.)
<code>math.h</code>	Funzioni matematiche (funzioni <code>sin()</code> , <code>cos()</code> , <code>sqrt()</code> , etc.)
<code>time.h</code>	Gestione data e ora
<code>ctype.h</code>	Classificazione caratteri (funzioni <code>isdigit()</code> , <code>isalpha()</code> , etc.)

Tabella 1.1: Principali librerie standard del C

### 1.10 Best Practices

1. **Indenta il codice:** Usa 4 spazi o un tab per ogni livello di indentazione

2. **Commenta:** Spiega cosa fa il codice, soprattutto le parti complesse
3. **Nomi significativi:** Usa nomi descrittivi per variabili e funzioni
4. **Una funzionalità per riga:** Evita di mettere troppe operazioni su una singola riga
5. **Compila con warning:** Usa sempre `-Wall -Wextra`

## 1.11 Esercizi

### 1.11.1 Livello Base

1. Scrivi un programma che stampa il tuo nome e cognome su righe separate.
2. Crea un programma che stampa un rettangolo di asterischi (5 righe x 10 colonne).
3. Scrivi un programma che stampa la scritta "C Programming" usando caratteri ASCII art.

### 1.11.2 Livello Intermedio

1. Scrivi un programma che stampa i primi 10 numeri (da 1 a 10) usando 10 printf separate.
2. Crea un programma che stampa una piccola storia su più righe, usando tabulazioni per l'indentazione dei dialoghi.
3. Scrivi un programma che stampa una tabella formattata con informazioni personali (nome, età, città).

### 1.11.3 Livello Avanzato

1. Ricerca online come usare i codici ANSI per colorare il testo nel terminale e scrivi un programma che stampa testo colorato.
2. Scrivi un programma che stampa un menu di un'applicazione (simulato) con bordi e formattazione.

## 1.12 Riepilogo

In questo capitolo abbiamo imparato:

- La storia e le caratteristiche del linguaggio C
- La differenza tra compilazione e interpretazione
- Come installare e configurare l'ambiente di sviluppo
- La struttura base di un programma C
- Come scrivere, compilare ed eseguire il primo programma
- Gli errori comuni e come risolverli
- Le principali librerie standard

## 1.13 Approfondimenti

Per approfondire gli argomenti di questo capitolo:

- **Libro:** "The C Programming Language" - Kernighan & Ritchie (capitolo 1)
- **Online:** C Tutorial su Learn-C.org
- **Video:** Playlist "C Programming for Beginners" su YouTube
- **Documentazione:** GCC Manual (<https://gcc.gnu.org/onlinedocs/>)

Nel [Capitolo 2](#) inizieremo a lavorare con variabili e tipi di dati, rendendo i nostri programmi interattivi!

# Capitolo 2

## Variabili e Tipi di Dati

### 2.1 Introduzione

Le variabili sono uno dei concetti fondamentali della programmazione. Una **variabile** è un contenitore che memorizza un valore che può cambiare durante l'esecuzione del programma. Pensate a una variabile come a una scatola etichettata dove potete conservare informazioni.

### 2.2 Obiettivi di Apprendimento

Alla fine di questo capitolo sarai in grado di:

- Comprendere il concetto di variabile
- Dichiarare e inizializzare variabili
- Conoscere i tipi di dati base del C
- Usare gli specificatori di formato della funzione printf()
- Leggere input dall'utente con la funzione scanf() della libreria stdio
- Lavorare con costanti

### 2.3 Concetto di Variabile

Una variabile ha tre caratteristiche fondamentali:

**Nome** Identificatore univoco (es. `eta`, `altezza`)

**Tipo** Determina quali valori può contenere (es. intero, decimale)

**Valore** Il dato effettivamente memorizzato

```
1 int eta = 17;  
2 // nome della variabile: eta  
3 // tipo della variabile: int (intero)  
4 // valore della variabile: 17
```

Listing 2.1: Esempio concettuale di variabile

## 2.4 Tipi di Dati Base

Il C è un linguaggio **fortemente tipizzato**: ogni variabile deve avere un tipo dichiarato esplicitamente.

### 2.4.1 Tipi Interi

I tipi interi memorizzano numeri senza parte decimale.

Tipo	Dimensione	Range (tipico)	Formato
char	1 byte	-128 a 127	%c
unsigned char	1 byte	0 a 255	%c
short	2 byte	-32,768 a 32,767	%hd
unsigned short	2 byte	0 a 65,535	%hu
int	4 byte	-2.147.483.648 a 2.147.483.647	%d
unsigned int	4 byte	0 a 4.294.967.295	%u
long	4-8 byte	Dipende dalla piattaforma	%ld
unsigned long	4-8 byte	Dipende dalla piattaforma	%lu
long long	8 byte	Molto ampio	%lld

Tabella 2.1: Tipi interi in C

#### Nota

La parola chiave **unsigned** indica che la variabile può contenere solo valori positivi (senza segno). Questo raddoppia il valore massimo rappresentabile.

### 2.4.2 Tipi a Virgola Mobile

I tipi a virgola mobile memorizzano numeri con parte decimale.

Tipo	Dimensione	Precisione	Formato
float	4 byte	6-7 cifre decimali	%f
double	8 byte	15-16 cifre decimali	%lf
long double	10-16 byte	18-19 cifre decimali	%Lf

Tabella 2.2: Tipi a virgola mobile in C

### 2.4.3 Tipo Carattere

```

1 char lettera = 'A';      // La variabile lettera contiene un singolo
   carattere
2 char simbolo = '#';     // La variabile simbolo contiene il carattere
   '#'
3 char numero = '5';      // La variabile numero contiene il carattere
   '5', non il numero 5!
```

Listing 2.2: Tipo char

**Attenzione**

Un carattere deve essere racchiuso tra apici singoli 'A', mentre una stringa usa apici doppi "Ciao". Non confonderli!

## 2.5 Dichiarazione di Variabili

### 2.5.1 Sintassi Base

```
1 tipo nome_variabile;
```

Listing 2.3: Sintassi di dichiarazione

### 2.5.2 Esempi di Dichiarazione

```
1 int eta; // Dichiarazione semplice
2 float altezza;
3 char iniziale;
4
5 int x, y, z; // Dichiarazione multipla
6 float base, altezza, area;
```

Listing 2.4: Dichiarazione di variabili

### 2.5.3 Inizializzazione

**Inizializzare** una variabile significa assegnarle un valore iniziale al momento della dichiarazione.

```
1 int eta = 17; // Dichiarazione + inizializzazione
    della variabile eta
2 float altezza = 1.75; // Dichiarazione + inizializzazione
    della variabile altezza
3 char grado = 'A'; // Dichiarazione + inizializzazione
    della variabile grado
4
5 // Si puo' anche fare in due passaggi
6 int numero; // Dichiarazione della variabile
    numero
7 numero = 42; // Assegnamento del valore 42 alla
    variabile numero
```

Listing 2.5: Inizializzazione di variabili

**Nota**

È buona pratica inizializzare sempre le variabili. Una variabile non inizializzata contiene un valore casuale (quello che c'era in memoria prima).

## 2.6 Regole per i Nomi delle Variabili

I nomi delle variabili (identificatori) devono seguire queste regole:

### 2.6.1 Regole Obbligatorie

1. Possono contenere lettere (a-z, A-Z), cifre (0-9) e underscore (\_)
2. Devono iniziare con una lettera o underscore (non con una cifra)
3. Non possono essere parole chiave del C (es. `int`, `return`, `if`)
4. Sono case-sensitive: `eta`, `Eta` e `ETA` sono variabili diverse

### 2.6.2 Convenzioni Consigliate

1. Usa nomi significativi: `eta` invece di `x`
2. Per nomi composti usa snake\_case: `numero_studenti`
3. Usa nomi brevi ma descrittivi
4. Evita nomi di una sola lettera (tranne per contatori come `i`, `j`, `k`)

```
1 // VALIDI
2 int eta;
3 int numero_studenti;
4 float _temp;
5 char scelta1;
6
7 // NON VALIDI
8 int lnumero;           // Inizia con una cifra
9 int for;               // Parola chiave
10 int numero-studenti;  // Contiene trattino
```

Listing 2.6: Esempi di nomi validi e non

## 2.7 L'operatore sizeof

L'operatore `sizeof` del linguaggio C restituisce la dimensione in byte di un tipo di dato o di una variabile.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     printf("Dimensione di int: %lu byte\n", sizeof(int));
5     printf("Dimensione di float: %lu byte\n", sizeof(float));
6     printf("Dimensione di double: %lu byte\n", sizeof(double));
7     printf("Dimensione di char: %lu byte\n", sizeof(char));
8
9     int x;
10    printf("Dimensione della variabile x: %lu byte\n", sizeof(x));
11
12    return 0;
13 }
```

Listing 2.7: Uso di `sizeof`

Output tipico:

Dimensione di int: 4 byte  
 Dimensione di float: 4 byte  
 Dimensione di double: 8 byte  
 Dimensione di char: 1 byte  
 Dimensione di x: 4 byte

### Nota

Le dimensioni possono variare in base all'architettura del sistema (32-bit vs 64-bit) e al compilatore.

## 2.8 Specificatori di Formato per printf

Gli specificatori di formato indicano alla funzione `printf()` della libreria `stdio` come stampare un valore.

Specificatore	Tipo	Descrizione
%d o %i	int	Intero decimale con segno
%u	unsigned int	Intero decimale senza segno
%f	float/double	Numero a virgola mobile
%lf	double (scanf)	Double (per input)
%c	char	Singolo carattere
%s	char*	Stringa
%x	int	Intero in esadecimale
%o	int	Intero in ottale
%p	puntatore	Indirizzo di memoria
%%	-	Stampa il simbolo %

Tabella 2.3: Principali specificatori di formato

### 2.8.1 Formattazione Avanzata

```

1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int numero = 42;
5     float pi = 3.14159;
6
7     printf("Numero: %d\n", numero);           // Stampa il valore
8     printf("Con padding: %5d\n", numero);      della variabile numero: 42
9     printf("Con zero: %05d\n", numero);        // Stampa il valore
10    printf("Float: %f\n", pi);                  della variabile numero con padding: 42 (5 caratteri)
11    printf("2 decimali: %.2f\n", pi);           // Stampa il valore
12    printf("6 decimali: %.6f\n", pi);           della variabile numero con zeri: 00042
13    printf("2 decimali: %.2f\n", pi);           // Stampa il valore
14    printf("6 decimali: %.6f\n", pi);           della variabile pi: 3.141590
15    printf("2 decimali: %.2f\n", pi);           // Stampa il valore
16    printf("6 decimali: %.6f\n", pi);           della variabile pi con 2 decimali: 3.14
17    printf("6 decimali: %.6f\n", pi);           // Stampa il valore
18    printf("6 decimali: %.6f\n", pi);           della variabile pi con 6 decimali: 3.141590
19 }
```

```
15     return 0;
16 }
```

Listing 2.8: Controllo del formato di stampa

## 2.9 Input da Tastiera: scanf

La funzione `scanf()` della libreria `stdio` legge input dall'utente.

### 2.9.1 Sintassi Base

```
1 scanf("specificatore", &variabile);
```

Listing 2.9: Sintassi di scanf

#### Attenzione

Nota l'operatore `&` (ampersand) prima del nome della variabile! Questo operatore indica alla funzione `scanf()` l'indirizzo di memoria della variabile dove salvare il valore letto. Dimenticarlo è uno degli errori più comuni.

### 2.9.2 Esempi di Input

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int eta;
5     float altezza;
6     char iniziale;
7
8     printf("Inserisci la tua eta': ");
9     scanf("%d", &eta); // Legge un intero e lo salva nella variabile
                        eta
10
11     printf("Inserisci la tua altezza (in metri): ");
12     scanf("%f", &altezza); // Legge un float e lo salva nella
                            variabile altezza
13
14     printf("Inserisci l'iniziale del tuo nome: ");
15     scanf(" %c", &iniziale); // Legge un char e lo salva nella
                            variabile iniziale (nota lo spazio prima di %c)
16
17     printf("\nRiepilogo:\n");
18     printf("Eta': %d anni\n", eta); // Stampa il valore della
                            variabile eta
19     printf("Altezza: %.2f m\n", altezza); // Stampa il valore della
                            variabile altezza
20     printf("Iniziale: %c\n", iniziale); // Stampa il valore della
                            variabile iniziale
21
22     return 0;
23 }
```

Listing 2.10: Leggere input con scanf

**Errore Comune**

Quando si legge un carattere con la funzione `scanf()` dopo aver letto numeri, è necessario aggiungere uno spazio prima di `%c` per consumare eventuali caratteri di whitespace rimasti nel buffer di input:

```
1 scanf(" %c", &carattere); // Lo spazio prima di %c e' importante!
```

## 2.10 Costanti

Le **costanti** sono valori che non possono essere modificati durante l'esecuzione del programma.

### 2.10.1 Costanti con `#define`

```
1 #include <stdio.h>
2
3 #define PI 3.14159
4 #define MAX_STUDENTI 30
5 #define SALUTO "Ciao a tutti!"
6
7 int main(int argc, char** argv) {
8     float raggio = 5.0;
9     float area = PI * raggio * raggio; // Calcola l'area usando la
10        costante PI e la variabile raggio
11
12     printf("Area del cerchio: %.2f\n", area); // Stampa il valore
13        della variabile area
14     printf("Numero massimo studenti: %d\n", MAX_STUDENTI); // Stampa
15        il valore della costante MAX_STUDENTI
16     printf("%s\n", SALUTO); // Stampa il valore della costante SALUTO
17
18     // PI = 3.14; // ERRORE: non si puo' modificare il valore della
19        costante PI
20
21     return 0;
22 }
```

Listing 2.11: Costanti con `#define`

**Nota**

Per convenzione, i nomi delle costanti si scrivono in MAIUSCOLO con underscore.

### 2.10.2 Costanti con `const`

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     const float PI = 3.14159;
5     const int GIORNI_SETTIMANA = 7;
6 }
```

```

7  float raggio = 3.0;
8  float area = PI * raggio * raggio; // Calcola l'area usando la
   costante PI e la variabile raggio
9
10 printf("Area: %.2f\n", area); // Stampa il valore della variabile
   area
11 printf("Giorni in una settimana: %d\n", GIORNI_SETTIMANA); //
   Stampa il valore della costante GIORNI_SETTIMANA
12
13 // PI = 3.14; // ERRORE: la costante PI non e' modificabile
14
15 return 0;
16 }

```

Listing 2.12: Costanti con const

### 2.10.3 Differenza tra #define e const

Aspetto	#define	const
Elaborazione	Preprocessore (sostituzione testuale)	Compilatore (variabile vera)
Tipo	No tipo esplicito	Ha un tipo esplicito
Debug	Più difficile	Più facile
Scope	Globale (dal punto di definizione)	Rispetta lo scope

Tabella 2.4: Confronto tra #define e const

## 2.11 Conversioni di Tipo

### 2.11.1 Conversione Implicita

Il C converte automaticamente i tipi quando necessario:

```

1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      int intero = 10;
5      float decimale = 3.5;
6
7      float risultato = intero + decimale; // Il valore della variabile
   intero viene convertito automaticamente in float
8      printf("Risultato: %.2f\n", risultato); // Stampa il valore della
   variabile risultato: 13.50
9
10     int troncato = intero + decimale; // Il risultato float viene
   troncato e assegnato alla variabile troncato
11     printf("Troncato: %d\n", troncato); // Stampa il valore della
   variabile troncato: 13
12
13     return 0;
14 }

```

Listing 2.13: Conversione implicita

**Attenzione**

Quando si assegna un float a un int, la parte decimale viene **troncata** (tagliata via), non arrotondata!

### 2.11.2 Conversione Esplicita (Cast)

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int a = 7, b = 2;
5
6     // Divisione intera
7     float ris1 = a / b; // Le variabili a e b sono int, quindi la
8                          // divisione e' intera
9     printf("Senza cast: %.2f\n", ris1); // Stampa il valore della
10                                         // variabile ris1: 3.00 (divisione intera!)
11
12     // Con cast
13     float ris2 = (float)a / b; // Il valore della variabile a viene
14                                // convertito in float prima della divisione
15     printf("Con cast: %.2f\n", ris2); // Stampa il valore della
16                                         // variabile ris2: 3.50 (divisione float)
17
18     // Cast per arrotondamento
19     float x = 3.7;
20     int arrotondato = (int)(x + 0.5); // Il valore della variabile x
21                                         // viene arrotondato
22     printf("Arrotondato: %d\n", arrotondato); // Stampa il valore
23                                         // della variabile arrotondato: 4
24
25     return 0;
26 }
```

Listing 2.14: Cast esplicito

## 2.12 Esempi Pratici Completi

### 2.12.1 Esempio 1: Calcolo Area Rettangolo

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     float base, altezza;
5     float area, perimetro;
6
7     printf("=== CALCOLO AREA RETTANGOLO ===\n\n");
8
9     printf("Inserisci la base: ");
10    scanf("%f", &base);
11
12    printf("Inserisci l'altezza: ");
13    scanf("%f", &altezza);
14 }
```

```
15     area = base * altezza;
16     perimetro = 2 * (base + altezza);
17
18     printf("\nRisultati:\n");
19     printf("Area: %.2f\n", area);
20     printf("Perimetro: %.2f\n", perimetro);
21
22     return 0;
23 }
```

Listing 2.15: Calcolo area e perimetro di un rettangolo

### 2.12.2 Esempio 2: Conversione Temperature

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      float celsius, fahrenheit;
5
6      printf("Inserisci la temperatura in Celsius: ");
7      scanf("%f", &celsius);
8
9      fahrenheit = (celsius * 9.0 / 5.0) + 32.0;
10
11     printf("%.2f gradi Celsius = %.2f gradi Fahrenheit\n",
12            celsius, fahrenheit);
13
14     return 0;
15 }
```

Listing 2.16: Conversione da Celsius a Fahrenheit

### 2.12.3 Esempio 3: Calcolo Media

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      float num1, num2, num3;
5      float somma, media;
6
7      printf("Inserisci tre numeri:\n");
8
9      printf("Numero 1: ");
10     scanf("%f", &num1);
11
12     printf("Numero 2: ");
13     scanf("%f", &num2);
14
15     printf("Numero 3: ");
16     scanf("%f", &num3);
17
18     somma = num1 + num2 + num3;
19     media = somma / 3.0;
20
21     printf("\nSomma: %.2f\n", somma);
```

```
22     printf("Media: %.2f\n", media);
23
24     return 0;
25 }
```

Listing 2.17: Calcolo media di tre numeri

## 2.13 Overflow e Underflow

### 2.13.1 Overflow

Si verifica quando un valore supera il massimo rappresentabile:

```
1  #include <stdio.h>
2  #include <limits.h>
3
4  int main(int argc, char** argv) {
5      int massimo = INT_MAX; // Valore massimo per int
6      printf("Valore massimo: %d\n", massimo);
7
8      int overflow = massimo + 1;
9      printf("Dopo overflow: %d\n", overflow); // Numero negativo!
10
11     return 0;
12 }
```

Listing 2.18: Esempio di overflow

### 2.13.2 Underflow

Si verifica quando un numero è troppo piccolo per essere rappresentato:

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      float piccolo = 0.0000001;
5      float piccolissimo = piccolo / 10000000000;
6
7      printf("Numero piccolo: %f\n", piccolo);
8      printf("Numero piccolissimo: %f\n", piccolissimo); // Potrebbe
9      essere 0
10
11     return 0;
12 }
```

Listing 2.19: Esempio di underflow

## 2.14 Libreria limits.h

La libreria `limits.h` definisce le costanti per i valori limite dei tipi:

```
1  #include <stdio.h>
2  #include <limits.h>
```

```
3
4 int main(int argc, char** argv) {
5     printf("Range di char: %d a %d\n", CHAR_MIN, CHAR_MAX);
6     printf("Range di short: %d a %d\n", SHRT_MIN, SHRT_MAX);
7     printf("Range di int: %d a %d\n", INT_MIN, INT_MAX);
8     printf("Range di long: %ld a %ld\n", LONG_MIN, LONG_MAX);
9
10    return 0;
11 }
```

Listing 2.20: Uso di limits.h

## 2.15 Esercizi

### 2.15.1 Livello Base

1. Scrivi un programma che chiede nome ed età all'utente e li stampa.
2. Crea un programma che legge due numeri interi e stampa la loro somma, differenza, prodotto e quoziente.
3. Scrivi un programma che converte un valore in euro in dollari (tasso di cambio fisso: 1 euro = 1.10 dollari).
4. Crea un programma che calcola l'area di un cerchio dato il raggio (usa `#define` per  $\pi$ ).

### 2.15.2 Livello Intermedio

1. Scrivi un programma che calcola l'IMC (Indice di Massa Corporea):  $IMC = \text{peso} / (\text{altezza} * \text{altezza})$ .
2. Crea un programma che converte un numero di secondi in ore, minuti e secondi.
3. Scrivi un programma che calcola il perimetro e l'area di un triangolo rettangolo dati i due cateti (usa il teorema di Pitagora per l'ipotenusa).
4. Crea un programma che simula una cassa: l'utente inserisce il prezzo, la quantità e viene calcolato il totale con IVA al 22%.

### 2.15.3 Livello Avanzato

1. Scrivi un programma che legge tre coefficienti (a, b, c) di un'equazione di secondo grado e calcola il discriminante ( $\Delta = b^2 - 4ac$ ).
2. Crea un programma che converte un numero decimale in binario (solo per numeri piccoli, es. 0-15).
3. Scrivi un programma che calcola l'interesse composto:  $M = C(1 + r)^n$  dove C è il capitale, r il tasso di interesse, n gli anni.

## 2.16 Riepilogo

In questo capitolo abbiamo imparato:

- Il concetto di variabile e tipo di dato
- I tipi base del C: interi, float, char
- Come dichiarare e inizializzare variabili
- Le regole per i nomi delle variabili
- Gli specificatori di formato per printf
- Come leggere input con scanf
- La differenza tra costanti (`#define` e `const`)
- Le conversioni di tipo implicite ed esplicite
- I problemi di overflow e underflow

## 2.17 Approfondimenti

- **Documentazione:** C Data Types Reference
- **Esercizi online:** HackerRank C Practice
- **Video:** "C Variables and Data Types" su YouTube

Nel prossimo capitolo vedremo gli operatori ed espressioni, che ci permetteranno di effettuare calcoli e manipolazioni più complesse!

# Capitolo 3

## Operatori ed Espressioni

### 3.1 Introduzione agli Operatori

Gli operatori sono simboli speciali che eseguono operazioni su uno o più operandi (valori o variabili). Il C fornisce un ricco insieme di operatori che permettono di manipolare i dati in modi diversi.

#### Nota

Il risultato di un'operazione ha sempre un tipo di dato. È importante comprendere come gli operatori interagiscono con i diversi tipi di dati.

### 3.2 Operatori Aritmetici

Gli operatori aritmetici eseguono operazioni matematiche di base.

#### 3.2.1 Operatori Binari

```
1 // Operatori aritmetici base
2 int a = 10, b = 3;
3 int somma = a + b;      // Addizione: 13
4 int differenza = a - b;  // Sottrazione: 7
5 int prodotto = a * b;    // Moltiplicazione: 30
6 int quoziente = a / b;   // Divisione intera: 3
7 int resto = a % b;       // Modulo (resto): 1
8
9 // Divisione con float
10 float x = 10.0, y = 3.0;
11 float divisione = x / y; // Divisione float: 3.333...
```

#### Attenzione

La divisione tra interi produce sempre un risultato intero, troncando la parte decimale. Per ottenere un risultato float, almeno uno degli operandi deve essere float.

#### 3.2.2 Operatori Unari

```

1 int n = 5;
2 int positivo = +n;    // Segno positivo: 5
3 int negativo = -n;    // Segno negativo: -5
4
5 // Incremento e decremento
6 int x = 10;
7 x++;    // Post-incremento: prima usa x, poi incrementa
8 ++x;    // Pre-incremento: prima incrementa, poi usa x
9 x--;    // Post-decremento
10 --x;    // Pre-decremento

```

### 3.2.3 Differenza tra Pre e Post Incremento

```

1 int a = 5, b = 5;
2 int x = a++;    // x = 5, a = 6 (prima assegna, poi incrementa)
3 int y = ++b;    // y = 6, b = 6 (prima incrementa, poi assegna)
4
5 printf("x = %d, a = %d\n", x, a);    // x = 5, a = 6
6 printf("y = %d, b = %d\n", y, b);    // y = 6, b = 6

```

## 3.3 Operatori di Assegnamento

### 3.3.1 Assegnamento Semplice

```

1 int x = 10;    // Assegna 10 a x

```

### 3.3.2 Operatori di Assegnamento Composto

```

1 int x = 10;
2 x += 5;    // Equivalente a: x = x + 5; (x diventa 15)
3 x -= 3;    // Equivalente a: x = x - 3; (x diventa 12)
4 x *= 2;    // Equivalente a: x = x * 2; (x diventa 24)
5 x /= 4;    // Equivalente a: x = x / 4; (x diventa 6)
6 x %= 5;    // Equivalente a: x = x % 5; (x diventa 1)

```

## 3.4 Operatori Relazionali

Gli operatori relazionali confrontano due valori e restituiscono 1 (vero) o 0 (falso).

```

1 int a = 10, b = 20;
2
3 // Operatori di confronto
4 int uguale = (a == b);    // 0 (falso)
5 int diverso = (a != b);    // 1 (vero)
6 int maggiore = (a > b);    // 0 (falso)
7 int minore = (a < b);    // 1 (vero)
8 int maggiore_uguale = (a >= b);    // 0 (falso)
9 int minore_uguale = (a <= b);    // 1 (vero)

```

**Attenzione**

Non confondere l'operatore di uguaglianza `==` con l'operatore di assegnamento `=`. Questo è uno degli errori più comuni in C!

## 3.5 Operatori Logici

Gli operatori logici operano su valori booleani (0 = falso, diverso da 0 = vero).

```
1 int a = 1, b = 0; // 1 = vero, 0 = falso
2
3 // AND logico (&&): vero solo se entrambi sono veri
4 int and_result = a && b; // 0 (falso)
5
6 // OR logico (||): vero se almeno uno è vero
7 int or_result = a || b; // 1 (vero)
8
9 // NOT logico (!): inverte il valore
10 int not_a = !a; // 0 (falso)
11 int not_b = !b; // 1 (vero)
```

### 3.5.1 Valutazione in Cortocircuito

```
1 int x = 5, y = 0;
2
3 // Con &&: se il primo è falso, il secondo non viene valutato
4 if (y != 0 && x / y > 2) {
5     printf("Questa riga non viene stampata\n");
6 }
7
8 // Con ||: se il primo è vero, il secondo non viene valutato
9 if (x > 0 || y / x > 0) {
10     printf("Questa riga viene stampata\n");
11 }
```

**Nota**

La valutazione in cortocircuito è utile per evitare errori come la divisione per zero.

## 3.6 Operatori Bit a Bit

Gli operatori bit a bit operano direttamente sui singoli bit dei numeri.

```
1 unsigned int a = 12; // In binario: 1100
2 unsigned int b = 10; // In binario: 1010
3
4 // AND bit a bit
5 unsigned int and_bit = a & b; // 1000 = 8
6
7 // OR bit a bit
8 unsigned int or_bit = a | b; // 1110 = 14
9
```

```
10 // XOR bit a bit
11 unsigned int xor_bit = a ^ b; // 0110 = 6
12
13 // NOT bit a bit
14 unsigned int not_a = ~a; // 0011 (complemento)
15
16 // Shift a sinistra
17 unsigned int left = a << 1; // 11000 = 24
18
19 // Shift a destra
20 unsigned int right = a >> 1; // 0110 = 6
```

## 3.7 Operatore Ternario

L'operatore ternario è una forma compatta di if-else.

### 3.7.1 Sintassi

```
1 condizione ? valore_se_vero : valore_se_falso
```

### 3.7.2 Esempi

```
1 int a = 10, b = 20;
2
3 // Trova il massimo
4 int max = (a > b) ? a : b;
5 printf("Il massimo e': %d\n", max); // 20
6
7 // Verifica parita'
8 char* parita = (a % 2 == 0) ? "pari" : "dispari";
9 printf("%d e' %s\n", a, parita); // 10 e' pari
10
11 // Assegnamento condizionale
12 int x = 5;
13 int y = (x > 0) ? x : -x; // Valore assoluto
```

## 3.8 Precedenza degli Operatori

La precedenza determina l'ordine in cui vengono eseguiti gli operatori in un'espressione.

### 3.8.1 Tabella di Precedenza (dal più alto al più basso)

1. Parentesi: ( )
2. Unari: ++ -- ! - + \* (dereferenziazione) & (indirizzo)
3. Moltiplicativi: \* / %
4. Additivi: + -

5. Shift: << >>
6. Relazionali: < <= > >=
7. Uguaglianza: == !=
8. AND bit a bit: &
9. XOR bit a bit: ^
10. OR bit a bit: |
11. AND logico: &&
12. OR logico: ||
13. Ternario: ? :
14. Assegnamento: = += -= \*= /= %= &= |= ^= <<= >>=

### 3.8.2 Esempi di Precedenza

```
1 int risultato;
2
3 // Esempio 1: moltiplicazione prima dell'addizione
4 risultato = 2 + 3 * 4; // 14, non 20
5
6 // Esempio 2: uso delle parentesi
7 risultato = (2 + 3) * 4; // 20
8
9 // Esempio 3: operatori logici e relazionali
10 int a = 5, b = 10, c = 15;
11 int test = a < b && b < c; // 1 (vero)
12
13 // Esempio 4: mix di operatori
14 risultato = a + b * c / 2 - 3; // 5 + (10*15)/2 - 3 = 77
```

#### Attenzione

Quando hai dubbi sulla precedenza, usa sempre le parentesi per rendere esplicito l'ordine di valutazione. Questo rende anche il codice più leggibile.

## 3.9 Conversioni di Tipo

### 3.9.1 Conversione Implicita (Casting Automatico)

```
1 int a = 10;
2 float b = 3.5;
3
4 // Promozione automatica: int -> float
5 float risultato = a + b; // 13.5
6 printf("Risultato: %.1f\n", risultato);
7
8 // Troncamento automatico: float -> int
9 int intero = a + b; // 13 (parte decimale troncata)
10 printf("Intero: %d\n", intero);
```

### 3.9.2 Conversione Esplicita (Cast)

```
1 int a = 10, b = 3;
2
3 // Senza cast: divisione intera
4 int div1 = a / b; // 3
5
6 // Con cast: divisione float
7 float div2 = (float)a / b; // 3.333...
8 float div3 = a / (float)b; // 3.333...
9 float div4 = (float)(a / b); // 3.0 (cast dopo la divisione)
10
11 printf("div1 = %d\n", div1);
12 printf("div2 = %.2f\n", div2);
13 printf("div3 = %.2f\n", div3);
14 printf("div4 = %.2f\n", div4);
```

#### Nota

La posizione del cast è importante: `(float)(a / b)` esegue prima la divisione intera e poi converte il risultato, mentre `(float)a / b` converte prima `a` in float e poi esegue la divisione float.

## 3.10 Espressioni Complesse

### 3.10.1 Combinazione di Operatori

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int a = 5, b = 10, c = 3;
5
6     // Espressione complessa
7     int risultato = (a + b) * c - (b / c) + a % c;
8     // (5 + 10) * 3 - (10 / 3) + 5 % 3
9     // 15 * 3 - 3 + 2
10    // 45 - 3 + 2
11    // 44
12
13    printf("Risultato: %d\n", risultato);
14
15    // Espressione con operatori logici
16    int x = 8;
17    if (x > 0 && x < 10 && x % 2 == 0) {
18        printf("%d e' un numero pari tra 1 e 9\n", x);
19    }
20
21    return 0;
22 }
```

### 3.10.2 Uso dell'Operatore sizeof

```
1 #include <stdio.h>
```

```
2
3 int main(int argc, char** argv) {
4     printf("Dimensione di int: %lu byte\n", sizeof(int));
5     printf("Dimensione di float: %lu byte\n", sizeof(float));
6     printf("Dimensione di double: %lu byte\n", sizeof(double));
7     printf("Dimensione di char: %lu byte\n", sizeof(char));
8
9     int array[10];
10    printf("Dimensione dell'array: %lu byte\n", sizeof(array));
11    printf("Numero di elementi: %lu\n", sizeof(array) / sizeof(int));
12
13    return 0;
14 }
```

## 3.11 Errori Comuni

### 3.11.1 Confusione tra = e ==

```
1 int x = 5;
2
3 // SBAGLIATO: assegnamento invece di confronto
4 if (x = 10) { // Assegna 10 a x e valuta come vero
5     printf("Questo viene sempre eseguito!\n");
6 }
7
8 // CORRETTO: confronto
9 if (x == 10) {
10     printf("x e' uguale a 10\n");
11 }
```

### 3.11.2 Divisione Intera Inaspettata

```
1 // SBAGLIATO: entrambi gli operandi sono interi
2 float media = (3 + 4 + 5) / 3; // 4.0 (non 4.0)
3
4 // CORRETTO: almeno un operando float
5 float media_corretta = (3 + 4 + 5) / 3.0; // 4.0
```

### 3.11.3 Overflow in Operazioni Aritmetiche

```
1 #include <stdio.h>
2 #include <limits.h>
3
4 int main(int argc, char** argv) {
5     int max = INT_MAX; // Valore massimo per int
6     printf("Valore massimo: %d\n", max);
7
8     // Overflow: il risultato supera il massimo
9     int overflow = max + 1;
10    printf("Dopo overflow: %d\n", overflow); // Numero negativo!
11
12    return 0;
```

13 }

## 3.12 Esercizi

### 3.12.1 Livello Base

1. Scrivi un programma che calcola l'area e il perimetro di un rettangolo dati base e altezza.
2. Crea un programma che converte una temperatura da Celsius a Fahrenheit usando la formula:  $F = C \times \frac{9}{5} + 32$ .
3. Scrivi un programma che verifica se un numero è pari o dispari usando l'operatore modulo.
4. Implementa un programma che calcola la media di tre numeri float.

### 3.12.2 Livello Intermedio

1. Scrivi un programma che dato un numero di secondi, calcola quante ore, minuti e secondi rappresenta.
2. Crea un programma che scambia i valori di due variabili senza usare una variabile temporanea (usa operatori aritmetici o XOR).
3. Implementa un programma che verifica se un anno è bisestile usando operatori logici e relazionali.
4. Scrivi un programma che estrae le singole cifre di un numero intero a tre cifre.

### 3.12.3 Livello Avanzato

1. Crea un programma che implementa una calcolatrice semplice con le quattro operazioni base, gestendo la divisione per zero.
2. Scrivi un programma che usa operatori bit a bit per verificare se un numero è una potenza di 2.
3. Implementa un programma che calcola il massimo comun divisore (MCD) di due numeri usando l'algoritmo di Euclide.
4. Crea un programma che dato un numero intero, conta quanti bit sono impostati a 1 nella sua rappresentazione binaria.

# Capitolo 4

## Controllo di Flusso

### 4.1 Introduzione

Il controllo di flusso permette di modificare l'ordine di esecuzione delle istruzioni in un programma. Senza strutture di controllo, le istruzioni verrebbero eseguite sequenzialmente dall'inizio alla fine. Le strutture di controllo permettono di:

- Prendere decisioni (istruzioni condizionali)
- Ripetere blocchi di codice (cicli)
- Saltare parti di codice

Il controllo di flusso è fondamentale per creare programmi dinamici e interattivi. Immagina di scrivere un programma che valuta i voti degli studenti: senza istruzioni condizionali, non potresti distinguere tra sufficiente e insufficiente. Oppure pensa a un programma che deve elaborare 100 numeri: senza cicli, dovresti scrivere la stessa istruzione 100 volte!

#### 4.1.1 Obiettivi di Apprendimento

Al termine di questo capitolo sarai in grado di:

- Utilizzare le istruzioni condizionali (`if`, `if-else`, `switch`) per prendere decisioni nel codice
- Implementare cicli (`while`, `do-while`, `for`) per ripetere operazioni
- Controllare il flusso dei cicli con `break` e `continue`
- Scegliere la struttura di controllo più appropriata per ogni situazione
- Evitare errori comuni come cicli infiniti e condizioni mal formulate
- Scrivere programmi complessi che combinano diverse strutture di controllo

### 4.2 Istruzioni Condizionali

#### 4.2.1 Istruzione `if`

L'istruzione `if` esegue un blocco di codice solo se una condizione è vera.

## Sintassi Base

```
1 if (condizione) {  
2     // codice eseguito se condizione e' vera  
3 }
```

## Esempi

```
1 #include <stdio.h>  
2  
3 int main(int argc, char** argv) {  
4     int eta = 18;  
5  
6     if (eta >= 18) {  
7         printf("Sei maggiorenne\n");  
8     }  
9  
10    // Esempio con espressione complessa  
11    int voto = 28;  
12    if (voto >= 18 && voto <= 30) {  
13        printf("Voto valido: %d\n", voto);  
14    }  
15  
16    return 0;  
17 }
```

### 4.2.2 Istruzione if-else

L'istruzione if-else permette di specificare un blocco alternativo da eseguire se la condizione è falsa.

## Sintassi

```
1 if (condizione) {  
2     // codice se condizione e' vera  
3 } else {  
4     // codice se condizione e' falsa  
5 }
```

## Esempi

```
1 #include <stdio.h>  
2  
3 int main(int argc, char** argv) {  
4     int numero = 7;  
5  
6     if (numero % 2 == 0) {  
7         printf("%d e' pari\n", numero);  
8     } else {  
9         printf("%d e' dispari\n", numero);  
10    }  
11 }
```

```
11 // Esempio con input
12 float temperatura;
13 printf("Inserisci la temperatura: ");
14 scanf("%f", &temperatura);
15
16 if (temperatura > 30) {
17     printf("Fa caldo!\n");
18 } else {
19     printf("La temperatura e' accettabile.\n");
20 }
21
22 return 0;
23 }
24
```

### 4.2.3 Istruzione if-else if-else

Per testare condizioni multiple in sequenza.

#### Sintassi

```
1 if (condizione1) {
2     // codice se condizione1 e' vera
3 } else if (condizione2) {
4     // codice se condizione2 e' vera
5 } else if (condizione3) {
6     // codice se condizione3 e' vera
7 } else {
8     // codice se nessuna condizione e' vera
9 }
```

#### Esempi

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int voto;
5     printf("Inserisci il voto: ");
6     scanf("%d", &voto);
7
8     if (voto < 18) {
9         printf("Insufficiente\n");
10    } else if (voto >= 18 && voto < 24) {
11        printf("Sufficiente\n");
12    } else if (voto >= 24 && voto < 27) {
13        printf("Buono\n");
14    } else if (voto >= 27 && voto <= 30) {
15        printf("Ottimo\n");
16    } else {
17        printf("Voto non valido\n");
18    }
19
20    return 0;
21 }
```

#### 4.2.4 if Annidati

È possibile inserire istruzioni `if` all'interno di altre istruzioni `if`.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int eta, patente;
5
6     printf("Inserisci eta': ");
7     scanf("%d", &eta);
8     printf("Hai la patente? (1=si, 0=no): ");
9     scanf("%d", &patente);
10
11     if (eta >= 18) {
12         if (patente == 1) {
13             printf("Puoi guidare!\n");
14         } else {
15             printf("Devi prendere la patente.\n");
16         }
17     } else {
18         printf("Sei troppo giovane per guidare.\n");
19     }
20
21     return 0;
22 }
```

##### Attenzione

Troppi livelli di annidamento rendono il codice difficile da leggere. Considera di usare operatori logici o di semplificare la logica.

#### 4.2.5 Istruzione switch

L'istruzione `switch` permette di confrontare una variabile con diversi valori costanti.

##### Sintassi

```
1 switch (espressione) {
2     case valore1:
3         // codice per valore1
4         break;
5     case valore2:
6         // codice per valore2
7         break;
8     default:
9         // codice se nessun case corrisponde
10 }
```

##### Esempi

```
1 #include <stdio.h>
2
```

```
3 int main(int argc, char** argv) {
4     int giorno;
5     printf("Inserisci un numero (1-7): ");
6     scanf("%d", &giorno);
7
8     switch (giorno) {
9         case 1:
10            printf("Lunedì\n");
11            break;
12        case 2:
13            printf("Martedì\n");
14            break;
15        case 3:
16            printf("Mercoledì\n");
17            break;
18        case 4:
19            printf("Giovedì\n");
20            break;
21        case 5:
22            printf("Venerdì\n");
23            break;
24        case 6:
25            printf("Sabato\n");
26            break;
27        case 7:
28            printf("Domenica\n");
29            break;
30        default:
31            printf("Giorno non valido\n");
32    }
33
34    return 0;
35 }
```

## Fall-through

Se ometti il `break`, l'esecuzione continua nel caso successivo.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int mese;
5     printf("Inserisci il numero del mese (1-12): ");
6     scanf("%d", &mese);
7
8     int giorni;
9     switch (mese) {
10        case 1: case 3: case 5: case 7:
11        case 8: case 10: case 12:
12            giorni = 31;
13            break;
14        case 4: case 6: case 9: case 11:
15            giorni = 30;
16            break;
17        case 2:
18            giorni = 28; // Semplificato, non considera anni
                           bisestili
19    }
```

```
19         break;
20     default:
21         printf("Mese non valido\n");
22         return 1;
23     }
24
25     printf("Il mese %d ha %d giorni\n", mese, giorni);
26     return 0;
27 }
```

#### Nota

Il **break** è essenziale nella maggior parte dei casi. Il fall-through può essere utile, ma deve essere usato intenzionalmente e documentato.

## 4.3 Cicli (Loop)

I cicli permettono di ripetere un blocco di codice più volte.

### 4.3.1 Ciclo while

Ripete un blocco di codice finché una condizione è vera. La condizione viene controllata all'inizio.

#### Sintassi

```
1 while (condizione) {
2     // codice da ripetere
3 }
```

#### Esempi

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     // Stampa numeri da 1 a 5
5     int i = 1;
6     while (i <= 5) {
7         printf("%d ", i);
8         i++;
9     }
10    printf("\n");
11
12    // Calcola somma di numeri inseriti dall'utente
13    int numero, somma = 0;
14    printf("Inserisci numeri (0 per terminare):\n");
15    scanf("%d", &numero);
16
17    while (numero != 0) {
18        somma += numero;
19        scanf("%d", &numero);
20    }
```

```
21
22     printf("La somma e': %d\n", somma);
23     return 0;
24 }
```

### 4.3.2 Ciclo do-while

Simile al `while`, ma la condizione viene controllata alla fine. Il blocco viene eseguito almeno una volta.

#### Sintassi

```
1 do {
2     // codice da ripetere
3 } while (condizione);
```

#### Esempi

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int numero;
5
6     // Richiede input finche' non e' valido
7     do {
8         printf("Inserisci un numero tra 1 e 10: ");
9         scanf("%d", &numero);
10
11         if (numero < 1 || numero > 10) {
12             printf("Numero non valido! Riprova.\n");
13         }
14     } while (numero < 1 || numero > 10);
15
16     printf("Hai inserito: %d\n", numero);
17
18     // Menu con do-while
19     int scelta;
20     do {
21         printf("\n=== MENU ===\n");
22         printf("1. Opzione 1\n");
23         printf("2. Opzione 2\n");
24         printf("3. Opzione 3\n");
25         printf("0. Esci\n");
26         printf("Scelta: ");
27         scanf("%d", &scelta);
28
29         switch (scelta) {
30             case 1:
31                 printf("Hai scelto l'opzione 1\n");
32                 break;
33             case 2:
34                 printf("Hai scelto l'opzione 2\n");
35                 break;
```

```
36         case 3:
37             printf("Hai scelto l'opzione 3\n");
38             break;
39         case 0:
40             printf("Uscita...\n");
41             break;
42         default:
43             printf("Scelta non valida\n");
44     }
45     } while (scelta != 0);
46
47     return 0;
48 }
```

#### Nota

Usa `do-while` quando vuoi che il codice venga eseguito almeno una volta, come nei menu o nella validazione dell'input.

### 4.3.3 Ciclo for

Il ciclo `for` è ideale quando si conosce in anticipo quante iterazioni fare.

#### Sintassi

```
1 for (inizializzazione; condizione; incremento) {
2     // codice da ripetere
3 }
```

#### Componenti del for

1. **Inizializzazione:** eseguita una sola volta all'inizio
2. **Condizione:** controllata prima di ogni iterazione
3. **Incremento:** eseguito dopo ogni iterazione

#### Esempi

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     // Stampa numeri da 1 a 10
5     for (int i = 1; i <= 10; i++) {
6         printf("%d ", i);
7     }
8     printf("\n");
9
10    // Stampa numeri pari da 2 a 20
11    for (int i = 2; i <= 20; i += 2) {
12        printf("%d ", i);
13    }
```

```
14     printf("\n");
15
16     // Conta alla rovescia
17     for (int i = 10; i >= 1; i--) {
18         printf("%d ", i);
19     }
20     printf("\n");
21
22     // Calcola fattoriale
23     int n = 5;
24     int fattoriale = 1;
25     for (int i = 1; i <= n; i++) {
26         fattoriale *= i;
27     }
28     printf("Il fattoriale di %d e': %d\n", n, fattoriale);
29
30     return 0;
31 }
```

### for Annidati

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      // Stampa una tabella pitagorica 5x5
5      printf("Tabella Pitagorica:\n");
6      for (int i = 1; i <= 5; i++) {
7          for (int j = 1; j <= 5; j++) {
8              printf("%4d", i * j);
9          }
10         printf("\n");
11     }
12
13     // Disegna un triangolo di asterischi
14     printf("\nTriangolo:\n");
15     for (int i = 1; i <= 5; i++) {
16         for (int j = 1; j <= i; j++) {
17             printf("*");
18         }
19         printf("\n");
20     }
21
22     return 0;
23 }
```

## 4.4 Istruzioni di Controllo dei Cicli

### 4.4.1 Istruzione break

L'istruzione `break` interrompe immediatamente il ciclo più interno.

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
```

```

4      // Trova il primo numero divisibile per 7
5      for (int i = 1; i <= 100; i++) {
6          if (i % 7 == 0) {
7              printf("Primo numero divisibile per 7: %d\n", i);
8              break; // Esce dal ciclo
9          }
10     }
11
12     // Esempio con while
13     int numero;
14     printf("Inserisci numeri (999 per uscire):\n");
15     while (1) { // Ciclo infinito
16         scanf("%d", &numero);
17         if (numero == 999) {
18             break; // Esce dal ciclo
19         }
20         printf("Hai inserito: %d\n", numero);
21     }
22
23     return 0;
24 }

```

#### 4.4.2 Istruzione continue

L'istruzione continue salta alla prossima iterazione del ciclo.

```

1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      // Stampa numeri da 1 a 10, saltando i multipli di 3
5      for (int i = 1; i <= 10; i++) {
6          if (i % 3 == 0) {
7              continue; // Salta questa iterazione
8          }
9          printf("%d ", i); // Non viene eseguito per 3, 6, 9
10     }
11     printf("\n");
12
13     // Calcola somma di numeri positivi
14     int numeri[] = {5, -3, 8, -1, 12, -7, 4};
15     int somma = 0;
16
17     for (int i = 0; i < 7; i++) {
18         if (numeri[i] < 0) {
19             continue; // Salta i numeri negativi
20         }
21         somma += numeri[i];
22     }
23
24     printf("Somma dei numeri positivi: %d\n", somma);
25     return 0;
26 }

```

**Attenzione**

`break` e `continue` devono essere usati con cautela: un uso eccessivo può rendere il codice difficile da seguire.

### 4.4.3 Istruzione goto

L'istruzione `goto` permette di saltare a un'etichetta specifica nel codice.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int i = 0;
5
6     inizio: // Etichetta
7     if (i < 5) {
8         printf("%d ", i);
9         i++;
10        goto inizio; // Salta all'etichetta
11    }
12
13    printf("\nFine\n");
14    return 0;
15 }
```

**Attenzione**

L'uso di `goto` è generalmente sconsigliato perché rende il codice difficile da comprendere e mantenere. Usa le strutture di controllo standard quando possibile.

## 4.5 Confronto tra i Cicli

### 4.5.1 Quando usare quale ciclo

- **for**: quando conosci il numero di iterazioni

```
1     for (int i = 0; i < 10; i++) {
2         // Esegui 10 volte
3     }
```

- **while**: quando la condizione deve essere controllata prima

```
1     while (numero != 0) {
2         // Potrebbe non eseguire mai
3     }
```

- **do-while**: quando vuoi eseguire almeno una volta

```
1     do {
2         // Esegue sempre almeno una volta
3     } while (condizione);
```

### 4.5.2 Equivalenza tra i Cicli

Questi tre cicli sono equivalenti:

```
1 // Ciclo for
2 for (int i = 0; i < 10; i++) {
3     printf("%d ", i);
4 }
5
6 // Equivalente con while
7 int i = 0;
8 while (i < 10) {
9     printf("%d ", i);
10    i++;
11 }
12
13 // Equivalente con do-while (se si esegue almeno una volta)
14 int i = 0;
15 if (i < 10) {
16     do {
17         printf("%d ", i);
18         i++;
19     } while (i < 10);
20 }
```

## 4.6 Esempi Completi

### 4.6.1 Calcolatrice con Menu

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     float num1, num2, risultato;
5     int scelta;
6
7     do {
8         printf("\n=== CALCOLATRICE ===\n");
9         printf("1. Addizione\n");
10        printf("2. Sottrazione\n");
11        printf("3. Moltiplicazione\n");
12        printf("4. Divisione\n");
13        printf("0. Esci\n");
14        printf("Scelta: ");
15        scanf("%d", &scelta);
16
17        if (scelta == 0) {
18            break;
19        }
20
21        if (scelta < 1 || scelta > 4) {
22            printf("Scelta non valida!\n");
23            continue;
24        }
25
26        printf("Inserisci primo numero: ");
27        scanf("%f", &num1);
```

```

28     printf("Inserisci secondo numero: ");
29     scanf("%f", &num2);
30
31     switch (scelta) {
32     case 1:
33         risultato = num1 + num2;
34         printf("%.2f + %.2f = %.2f\n", num1, num2, risultato);
35         break;
36     case 2:
37         risultato = num1 - num2;
38         printf("%.2f - %.2f = %.2f\n", num1, num2, risultato);
39         break;
40     case 3:
41         risultato = num1 * num2;
42         printf("%.2f * %.2f = %.2f\n", num1, num2, risultato);
43         break;
44     case 4:
45         if (num2 != 0) {
46             risultato = num1 / num2;
47             printf("%.2f / %.2f = %.2f\n",
48                 num1, num2, risultato);
49         } else {
50             printf("Errore: divisione per zero!\n");
51         }
52         break;
53     }
54 } while (1);
55
56 printf("Arrivederci!\n");
57 return 0;
58 }

```

#### 4.6.2 Gioco: Indovina il Numero

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(int argc, char** argv) {
6      int numero_segreto, tentativo, tentativi = 0;
7
8      // Inizializza il generatore di numeri casuali
9      srand(time(NULL));
10     numero_segreto = rand() % 100 + 1; // Numero tra 1 e 100
11
12     printf("=== INDOVINA IL NUMERO ===\n");
13     printf("Ho pensato un numero tra 1 e 100.\n");
14
15     do {
16         printf("\nInserisci il tuo tentativo: ");
17         scanf("%d", &tentativo);
18         tentativi++;
19
20         if (tentativo < numero_segreto) {
21             printf("Troppo basso! Riprova.\n");
22         } else if (tentativo > numero_segreto) {
23             printf("Troppo alto! Riprova.\n");

```

```
24     } else {  
25         printf("\nComplimenti! Hai indovinato!\n");  
26         printf("Numero di tentativi: %d\n", tentativi);  
27     }  
28 } while (tentativo != numero_segreto);  
29  
30 return 0;  
31 }
```

## 4.7 Esercizi

### 4.7.1 Livello Base

1. Scrivi un programma che determina se un numero è positivo, negativo o zero.
2. Crea un programma che stampa i numeri da 1 a 100.
3. Scrivi un programma che calcola la somma dei primi N numeri naturali.
4. Implementa un programma che stampa la tavola pitagorica di un numero dato.

### 4.7.2 Livello Intermedio

1. Scrivi un programma che verifica se un numero è primo.
2. Crea un programma che inverte un numero (es: 1234 diventa 4321).
3. Implementa un programma che calcola il massimo comun divisore (MCD) di due numeri usando l'algoritmo di Euclide.
4. Scrivi un programma che disegna un triangolo di numeri:

```
1  
12  
123  
1234  
12345
```

### 4.7.3 Livello Avanzato

1. Crea un programma che genera la sequenza di Fibonacci fino a N termini.
2. Scrivi un programma che verifica se un numero è un palindromo.
3. Implementa un programma che trova tutti i numeri perfetti minori di 1000 (un numero perfetto è uguale alla somma dei suoi divisori propri).
4. Crea un programma che disegna il triangolo di Floyd:

```
1  
2 3  
4 5 6  
7 8 9 10
```

# Capitolo 5

## Funzioni

### 5.1 Introduzione alle Funzioni

Le funzioni sono blocchi di codice riutilizzabili che eseguono un compito specifico. Permettono di:

- Organizzare il codice in modo modulare
- Evitare la duplicazione del codice
- Rendere il programma più leggibile e manutenibile
- Facilitare il testing e il debugging

#### Nota

Ogni programma C ha almeno una funzione: la funzione `main()`, che è il punto di ingresso del programma.

#### 5.1.1 Obiettivi di apprendimento

Al termine di questo capitolo sarai in grado di:

- Definire e dichiarare funzioni con diversi tipi di parametri e valori di ritorno
- Comprendere il meccanismo del passaggio per valore
- Utilizzare i prototipi di funzione per organizzare il codice
- Implementare funzioni ricorsive per risolvere problemi complessi
- Gestire correttamente lo scope e la visibilità delle variabili
- Applicare buone pratiche nella scrittura di funzioni modulari

## 5.2 Definizione di una Funzione

### 5.2.1 Sintassi

```
1 tipo_ritorno nome_funzione(parametri) {  
2     // corpo della funzione  
3     return valore; // se tipo_ritorno != void  
4 }
```

### 5.2.2 Componenti di una Funzione

1. **Tipo di ritorno:** il tipo di dato restituito dalla funzione
2. **Nome:** identificatore della funzione
3. **Parametri:** input della funzione (opzionali)
4. **Corpo:** blocco di codice da eseguire
5. **Return:** valore restituito (se non void)

### 5.2.3 Esempio Semplice

```
1 #include <stdio.h>  
2  
3 // Funzione senza parametri che stampa un messaggio di saluto  
4 void saluta() {  
5     printf("Ciao, benvenuto!\n");  
6 }  
7  
8 int main(int argc, char** argv) {  
9     saluta(); // Chiamata alla funzione  
10    saluta(); // Chiamata alla funzione (di nuovo)  
11    return 0;  
12 }
```

## 5.3 Parametri e Argomenti

### 5.3.1 Funzioni con Parametri

```
1 #include <stdio.h>  
2  
3 // Funzione con un parametro di tipo puntatore a caratteri  
4 void saluta_nome(char* nome) {  
5     printf("Ciao, %s!\n", nome);  
6 }  
7  
8 // Funzione con piu' parametri  
9 // Funzione che stampa la somma di due interi  
10 void stampa_somma(int a, int b) {  
11     int somma = a + b;  
12     printf("%d + %d = %d\n", a, b, somma);  
13 }
```

```
13 }
14
15 int main(int argc, char** argv) {
16     saluta_nome("Mario");
17     saluta_nome("Lucia");
18
19     stampa_somma(5, 3);
20     stampa_somma(10, 20);
21
22     return 0;
23 }
```

### 5.3.2 Passaggio per Valore

In C, gli argomenti sono passati per valore: la funzione riceve una copia del valore.

```
1 #include <stdio.h>
2
3 // Funzione che riceve un intero per valore
4 void incrementa(int x) {
5     x = x + 1; // Modifica solo la copia locale del parametro
6     printf("Dentro la funzione: x = %d\n", x);
7 }
8
9 int main(int argc, char** argv) {
10     int numero = 5;
11     printf("Prima: numero = %d\n", numero);
12
13     incrementa(numero);
14
15     printf("Dopo: numero = %d\n", numero); // Ancora 5!
16     return 0;
17 }
```

Output:

```
Prima: numero = 5
Dentro la funzione: x = 6
Dopo: numero = 5
```

#### Attenzione

Le modifiche ai parametri all'interno della funzione non influenzano le variabili originali nel chiamante. Per modificare una variabile passata come parametro, è necessario usare i puntatori (vedi [Capitolo 7](#)).

## 5.4 Valore di Ritorno

### 5.4.1 Funzioni che Restituiscono un Valore

```
1 #include <stdio.h>
2
3 // Funzione che calcola la somma di due interi
```

```
4 int somma(int a, int b) {
5     return a + b;
6 }
7
8 // Funzione che calcola il massimo tra due interi
9 int massimo(int a, int b) {
10     if (a > b) {
11         return a;
12     } else {
13         return b;
14     }
15 }
16
17 // Funzione che verifica se un numero intero e' pari
18 int is_pari(int n) {
19     return (n % 2 == 0); // Restituisce 1 (vero) o 0 (falso) al
20                           chiamante
21 }
22
23 int main(int argc, char** argv) {
24     int risultato = somma(5, 3);
25     printf("Somma: %d\n", risultato);
26
27     printf("Massimo tra 10 e 25: %d\n", massimo(10, 25));
28
29     if (is_pari(8)) {
30         printf("8 e' pari\n");
31     }
32
33     return 0;
34 }
```

### 5.4.2 Funzioni void

Le funzioni con tipo di ritorno void non restituiscono alcun valore al chiamante.

```
1 #include <stdio.h>
2
3 // Funzione che stampa una linea di caratteri '-'
4 void stampa_linea(int lunghezza) {
5     for (int i = 0; i < lunghezza; i++) {
6         printf("-");
7     }
8     printf("\n");
9 }
10
11 // Funzione che stampa un menu con opzioni
12 void stampa_menu() {
13     stampa_linea(20); // Chiama la funzione stampa_linea
14     printf("1. Opzione 1\n");
15     printf("2. Opzione 2\n");
16     printf("3. Esci\n");
17     stampa_linea(20); // Chiama la funzione stampa_linea
18 }
19
20 int main(int argc, char** argv) {
21     stampa_menu();
22 }
```

```
22     return 0;
23 }
```

## 5.5 Dichiarazione e Prototipo

### 5.5.1 Prototipo di Funzione

Il prototipo dichiara l'esistenza di una funzione prima della sua definizione.

```
1  #include <stdio.h>
2
3  // Prototipi (dichiarazioni)
4  int somma(int a, int b);
5  int prodotto(int a, int b);
6  void stampa_risultato(int valore);
7
8  int main(int argc, char** argv) {
9      int a = 5, b = 3;
10
11      int s = somma(a, b);
12      int p = prodotto(a, b);
13
14      stampa_risultato(s);
15      stampa_risultato(p);
16
17      return 0;
18 }
19
20 // Definizioni delle funzioni
21 int somma(int a, int b) {
22     return a + b;
23 }
24
25 int prodotto(int a, int b) {
26     return a * b;
27 }
28
29 void stampa_risultato(int valore) {
30     printf("Risultato: %d\n", valore);
31 }
```

#### Nota

I prototipi permettono di definire le funzioni in qualsiasi ordine e di separarle in file diversi.

## 5.6 Funzioni Ricorsive

Una funzione ricorsiva è una funzione che chiama se stessa.

### 5.6.1 Fattoriale Ricorsivo

```
1 #include <stdio.h>
2
3 // Funzione ricorsiva che calcola il fattoriale di un intero
4 int fattoriale(int n) {
5     // Caso base della ricorsione
6     if (n == 0 || n == 1) {
7         return 1;
8     }
9     // Caso ricorsivo: la funzione chiama se stessa
10    return n * fattoriale(n - 1);
11 }
12
13 int main(int argc, char** argv) {
14     int n = 5;
15     printf("Fattoriale di %d = %d\n", n, fattoriale(n));
16     return 0;
17 }
```

### 5.6.2 Fibonacci Ricorsivo

```
1 #include <stdio.h>
2
3 // Funzione ricorsiva che calcola l'n-esimo numero di Fibonacci
4 int fibonacci(int n) {
5     // Casi base della ricorsione
6     if (n == 0) {
7         return 0;
8     }
9     if (n == 1) {
10        return 1;
11    }
12    // Caso ricorsivo: la funzione chiama se stessa due volte
13    return fibonacci(n - 1) + fibonacci(n - 2);
14 }
15
16 int main(int argc, char** argv) {
17     printf("Sequenza di Fibonacci:\n");
18     for (int i = 0; i < 10; i++) {
19         printf("%d ", fibonacci(i));
20     }
21     printf("\n");
22     return 0;
23 }
```

#### Attenzione

Le funzioni ricorsive devono sempre avere un caso base per evitare la ricorsione infinita. Inoltre, la ricorsione può essere inefficiente per alcuni problemi.

## 5.7 Ambito delle Variabili (Scope)

### 5.7.1 Variabili Locali

Le variabili dichiarate all'interno di una funzione sono locali a quella funzione.

```
1 #include <stdio.h>
2
3 void funzione1() {
4     int x = 10; // Variabile locale alla funzione funzione1
5     printf("funzione1: x = %d\n", x);
6 }
7
8 void funzione2() {
9     int x = 20; // Diversa variabile locale, indipendente da quella
10    // in funzione1
11    printf("funzione2: x = %d\n", x);
12 }
13
14 int main(int argc, char** argv) {
15     int x = 5; // Variabile locale alla funzione main
16     printf("main: x = %d\n", x);
17
18     funzione1();
19     funzione2();
20
21     printf("main: x = %d\n", x); // Ancora 5
22     return 0;
23 }
```

### 5.7.2 Variabili Globali

Le variabili dichiarate fuori da tutte le funzioni sono globali e accessibili ovunque.

```
1 #include <stdio.h>
2
3 int contatore = 0; // Variabile globale
4
5 void incrementa() {
6     contatore++;
7     printf("Contatore: %d\n", contatore);
8 }
9
10 int main(int argc, char** argv) {
11     printf("Contatore iniziale: %d\n", contatore);
12
13     incrementa();
14     incrementa();
15     incrementa();
16
17     printf("Contatore finale: %d\n", contatore);
18     return 0;
19 }
```

**Attenzione**

L'uso eccessivo di variabili globali rende il codice difficile da gestire e debuggare. Preferisci passare i valori come parametri.

### 5.7.3 Variabili Static

Le variabili dichiarate con lo specificatore `static` mantengono il loro valore tra le diverse chiamate della funzione.

```
1 #include <stdio.h>
2
3 void conta_chiamate() {
4     static int contatore = 0; // Variabile static: inizializzata una
5                               sola volta
6     contatore++;
7     printf("Questa funzione e' stata chiamata %d volte\n",
8           contatore);
9 }
10
11 int main(int argc, char** argv) {
12     conta_chiamate(); // 1
13     conta_chiamate(); // 2
14     conta_chiamate(); // 3
15     return 0;
16 }
```

## 5.8 Esempi Pratici

### 5.8.1 Calcolatrice con Funzioni

```
1 #include <stdio.h>
2
3 // Funzione che calcola l'addizione di due numeri float
4 float addizione(float a, float b) {
5     return a + b;
6 }
7
8 // Funzione che calcola la sottrazione di due numeri float
9 float sottrazione(float a, float b) {
10     return a - b;
11 }
12
13 // Funzione che calcola la moltiplicazione di due numeri float
14 float moltiplicazione(float a, float b) {
15     return a * b;
16 }
17
18 // Funzione che calcola la divisione di due numeri float
19 float divisione(float a, float b) {
20     if (b != 0) {
21         return a / b;
22     } else {
23         printf("Errore: divisione per zero!\n");
24     }
25 }
```

```
24     return 0;
25 }
26 }
27
28 // Funzione che stampa il menu della calcolatrice
29 void stampa_menu() {
30     printf("\n=== CALCOLATRICE ===\n");
31     printf("1. Addizione\n");
32     printf("2. Sottrazione\n");
33     printf("3. Moltiplicazione\n");
34     printf("4. Divisione\n");
35     printf("0. Esci\n");
36 }
37
38 int main(int argc, char** argv) {
39     int scelta;
40     float num1, num2, risultato;
41
42     do {
43         stampa_menu();
44         printf("Scelta: ");
45         scanf("%d", &scelta);
46
47         if (scelta == 0) {
48             break;
49         }
50
51         if (scelta < 1 || scelta > 4) {
52             printf("Scelta non valida!\n");
53             continue;
54         }
55
56         printf("Primo numero: ");
57         scanf("%f", &num1);
58         printf("Secondo numero: ");
59         scanf("%f", &num2);
60
61         switch (scelta) {
62             case 1:
63                 risultato = addizione(num1, num2);
64                 printf("%.2f + %.2f = %.2f\n",
65                     num1, num2, risultato);
66                 break;
67             case 2:
68                 risultato = sottrazione(num1, num2);
69                 printf("%.2f - %.2f = %.2f\n",
70                     num1, num2, risultato);
71                 break;
72             case 3:
73                 risultato = moltiplicazione(num1, num2);
74                 printf("%.2f * %.2f = %.2f\n",
75                     num1, num2, risultato);
76                 break;
77             case 4:
78                 risultato = divisione(num1, num2);
79                 if (num2 != 0) {
80                     printf("%.2f / %.2f = %.2f\n",
81                         num1, num2, risultato);
```

```
82         }
83         break;
84     }
85 } while (1);
86
87 printf("Arrivederci!\n");
88 return 0;
89 }
```

## 5.8.2 Funzioni Matematiche

```
1  #include <stdio.h>
2
3  // Funzione che calcola la potenza intera (base^esponente)
4  int potenza(int base, int esponente) {
5      int risultato = 1;
6      for (int i = 0; i < esponente; i++) {
7          risultato *= base;
8      }
9      return risultato;
10 }
11
12 // Funzione che verifica se un intero e' un numero primo
13 int is_primo(int n) {
14     if (n <= 1) {
15         return 0;
16     }
17     for (int i = 2; i * i <= n; i++) {
18         if (n % i == 0) {
19             return 0;
20         }
21     }
22     return 1;
23 }
24
25 // Funzione che calcola il Massimo Comune Divisore usando l'algoritmo
    di Euclide
26 int mcd(int a, int b) {
27     while (b != 0) {
28         int temp = b;
29         b = a % b;
30         a = temp;
31     }
32     return a;
33 }
34
35 // Funzione che calcola il minimo comune multiplo di due interi
36 int mcm(int a, int b) {
37     return (a * b) / mcd(a, b); // Chiama la funzione mcd
38 }
39
40 int main(int argc, char** argv) {
41     printf("2^10 = %d\n", potenza(2, 10));
42     printf("17 e' primo? %s\n", is_primo(17) ? "Si" : "No");
43     printf("MCD(48, 18) = %d\n", mcd(48, 18));
44     printf("mcm(12, 18) = %d\n", mcm(12, 18));
45 }
```

```
46     return 0;
47 }
```

## 5.9 Buone Pratiche

### 5.9.1 Nomi Descrittivi

```
1  // SBAGLIATO: nomi poco chiari
2  int f(int a, int b) {
3      return a * b;
4  }
5
6  // CORRETTO: nomi descrittivi
7  int calcola_area Rettangolo(int base, int altezza) {
8      return base * altezza; // Restituisce l'area del rettangolo
9  }
```

### 5.9.2 Funzioni Piccole e Focalizzate

```
1  // Una funzione dovrebbe fare una sola cosa
2
3  // SBAGLIATO: troppo in una funzione
4  void gestisci_tutto(int scelta) {
5      // Lettura input
6      // Validazione
7      // Calcolo
8      // Stampa
9      // ...centinaia di righe...
10 }
11
12 // CORRETTO: dividi in funzioni piu' piccole
13 int leggi_scelta();
14 int valida_scelta(int scelta);
15 float esegui_calcolo(int scelta, float a, float b);
16 void stampa_risultato(float risultato);
```

### 5.9.3 Documentazione

```
1  /**
2   * Calcola il fattoriale di un numero intero.
3   *
4   * @param n Il numero intero di cui calcolare il fattoriale (n >= 0)
5   * @return Il fattoriale di n come intero
6   */
7  int fattoriale(int n) {
8      if (n == 0 || n == 1) {
9          return 1;
10     }
11     return n * fattoriale(n - 1); // Chiamata ricorsiva
12 }
```

## 5.10 Esercizi

### 5.10.1 Livello Base

1. Scrivi una funzione che converte una temperatura da Celsius a Fahrenheit.
2. Crea una funzione che verifica se un numero è pari.
3. Implementa una funzione che calcola l'area di un cerchio dato il raggio.
4. Scrivi una funzione che stampa un triangolo di asterischi di altezza N.

### 5.10.2 Livello Intermedio

1. Crea una funzione che inverte un numero intero (es:  $1234 \rightarrow 4321$ ).
2. Scrivi una funzione ricorsiva che calcola la somma delle cifre di un numero.
3. Implementa una funzione che verifica se una stringa è un palindromo.
4. Crea un set di funzioni per la conversione tra diverse unità di misura (km/miglia, kg/libbre, ecc.).

### 5.10.3 Livello Avanzato

1. Implementa le funzioni Torre di Hanoi in modo ricorsivo.
2. Scrivi una funzione che genera numeri primi fino a N usando il Crivello di Eratostene.
3. Crea una funzione ricorsiva per il calcolo del coefficiente binomiale.
4. Implementa una funzione che risolve il problema delle N regine usando la ricorsione e il backtracking.

# Capitolo 6

## Array

### 6.1 Introduzione agli Array

Un array è una struttura dati che contiene una collezione di elementi dello stesso tipo, memorizzati in locazioni di memoria contigue. Gli array sono fondamentali in programmazione perché permettono di gestire insiemi di dati correlati in modo efficiente, come liste di numeri, sequenze di caratteri, tabelle di valori e molto altro.

#### 6.1.1 Obiettivi di Apprendimento

Al termine di questo capitolo sarai in grado di:

- Dichiarare e inizializzare array monodimensionali e multidimensionali
- Accedere e modificare gli elementi di un array
- Iterare su array utilizzando cicli
- Passare array a funzioni e comprendere il passaggio per riferimento
- Implementare operazioni comuni sugli array (ricerca, ordinamento, inversione)
- Lavorare con matrici e operazioni bidimensionali
- Applicare algoritmi di ordinamento di base
- Evitare errori comuni nella gestione degli array

#### 6.1.2 Caratteristiche degli Array

- Tutti gli elementi hanno lo stesso tipo
- Gli elementi sono indicizzati a partire da 0
- La dimensione è fissa (definita alla creazione)
- Gli elementi sono memorizzati in memoria contigua

**Nota**

Gli array in C sono a base zero: il primo elemento ha indice 0, il secondo ha indice 1, e così via.

## 6.2 Dichiarazione e Inizializzazione

### 6.2.1 Sintassi Base

```
1 tipo nome_array[dimensione];
```

### 6.2.2 Esempi di Dichiarazione

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     // Dichiarazione di un array di 5 interi
5     int numeri[5];
6
7     // Inizializzazione esplicita
8     int valori[5] = {10, 20, 30, 40, 50};
9
10    // Inizializzazione parziale (gli altri elementi sono 0)
11    int parziale[5] = {1, 2}; // {1, 2, 0, 0, 0}
12
13    // Dimensione automatica
14    int auto_size[] = {5, 10, 15, 20}; // 4 elementi
15
16    // Inizializzazione a zero
17    int tutti_zero[100] = {0};
18
19    return 0;
20 }
```

## 6.3 Accesso agli Elementi

### 6.3.1 Lettura e Scrittura

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int numeri[5] = {10, 20, 30, 40, 50};
5
6     // Accesso in lettura
7     printf("Primo elemento: %d\n", numeri[0]); // 10
8     printf("Terzo elemento: %d\n", numeri[2]); // 30
9     printf("Ultimo elemento: %d\n", numeri[4]); // 50
10
11    // Accesso in scrittura
12    numeri[0] = 100;
13    numeri[2] = 300;
```

```
14     printf("Primo elemento modificato: %d\n", numeri[0]); // 100
15
16     return 0;
17 }
18 }
```

### Attenzione

Accedere a un indice fuori dai limiti dell'array causa comportamento indefinito. Il C non controlla automaticamente i limiti degli array!

## 6.3.2 Esempio di Errore

```
1 int numeri[5] = {1, 2, 3, 4, 5};
2
3 // SBAGLIATO: indice fuori dai limiti
4 numeri[5] = 100; // Errore! Gli indici validi sono 0-4
5 numeri[10] = 200; // Errore! Accesso a memoria non allocata
```

## 6.4 Attraversamento di un Array

### 6.4.1 Uso del Ciclo for

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int numeri[5] = {10, 20, 30, 40, 50};
5
6     // Stampa di tutti gli elementi
7     printf("Elementi dell'array:\n");
8     for (int i = 0; i < 5; i++) {
9         printf("numeri[%d] = %d\n", i, numeri[i]);
10    }
11
12    // Calcolo della somma degli elementi dell'array
13    int somma = 0;
14    for (int i = 0; i < 5; i++) {
15        somma += numeri[i];
16    }
17    printf("Somma: %d\n", somma);
18
19    // Calcolo della media degli elementi dell'array
20    float media = (float)somma / 5; // Cast a float per divisione
    decimale
21    printf("Media: %.2f\n", media);
22
23    return 0;
24 }
```

### 6.4.2 Dimensione Dinamica

```
1 #include <stdio.h>
2
3 #define DIM 10
4
5 int main(int argc, char** argv) {
6     int array[DIM];
7
8     // Inizializzazione dell'array con multipli di 2
9     for (int i = 0; i < DIM; i++) {
10         array[i] = i * 2;
11     }
12
13     // Stampa di tutti gli elementi dell'array
14     for (int i = 0; i < DIM; i++) {
15         printf("%d ", array[i]);
16     }
17     printf("\n");
18
19     return 0;
20 }
```

## 6.5 Input e Output di Array

### 6.5.1 Lettura da Tastiera

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int n;
5     printf("Quanti numeri vuoi inserire? ");
6     scanf("%d", &n);
7
8     int numeri[n]; // Array di interi di dimensione variabile (
9                     // standard C99)
10
11     // Lettura degli elementi dell'array da tastiera
12     printf("Inserisci %d numeri:\n", n);
13     for (int i = 0; i < n; i++) {
14         printf("Elemento %d: ", i + 1);
15         scanf("%d", &numeri[i]); // Legge l'i-esimo elemento
16     }
17
18     // Stampa degli elementi
19     printf("\nHai inserito:\n");
20     for (int i = 0; i < n; i++) {
21         printf("%d ", numeri[i]);
22     }
23     printf("\n");
24
25     return 0;
26 }
```

## 6.6 Operazioni sugli Array

### 6.6.1 Ricerca del Massimo e del Minimo

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int numeri[] = {45, 23, 78, 12, 67, 89, 34};
5     int n = sizeof(numeri) / sizeof(numeri[0]);
6
7     // Ricerca del massimo elemento nell'array
8     int max = numeri[0]; // Inizializza il massimo al primo elemento
9     for (int i = 1; i < n; i++) {
10         if (numeri[i] > max) {
11             max = numeri[i]; // Aggiorna il massimo se trovato
12                                 elemento maggiore
13         }
14     }
15     printf("Massimo: %d\n", max);
16
17     // Ricerca del minimo elemento nell'array
18     int min = numeri[0]; // Inizializza il minimo al primo elemento
19     for (int i = 1; i < n; i++) {
20         if (numeri[i] < min) {
21             min = numeri[i]; // Aggiorna il minimo se trovato
22                                 elemento minore
23         }
24     }
25     printf("Minimo: %d\n", min);
26
27     return 0;
28 }
```

### 6.6.2 Ricerca di un Elemento

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int numeri[] = {10, 25, 30, 45, 50, 60, 75};
5     int n = sizeof(numeri) / sizeof(numeri[0]); // Calcola la
6                                             dimensione dell'array
7     int da_cercare = 45;
8     int trovato = 0;
9     int posizione = -1;
10
11     // Ricerca lineare dell'elemento da_cercare nell'array
12     for (int i = 0; i < n; i++) {
13         if (numeri[i] == da_cercare) {
14             trovato = 1; // Elemento trovato
15             posizione = i; // Salva la posizione dell'elemento
16             break;
17         }
18     }
19
20     if (trovato) {
21         printf("%d trovato alla posizione %d\n",
22 
```

```
21         da_cercare, posizione);
22     } else {
23         printf("%d non trovato\n", da_cercare);
24     }
25
26     return 0;
27 }
```

### 6.6.3 Inversione di un Array

```
1  #include <stdio.h>
2
3  // Funzione che stampa tutti gli elementi di un array di interi
4  void stampa_array(int* arr, int n) {
5      for (int i = 0; i < n; i++) {
6          printf("%d ", arr[i]);
7      }
8      printf("\n");
9  }
10
11 // Funzione che inverte un array di interi sul posto
12 void inverti_array(int* arr, int n) {
13     for (int i = 0; i < n / 2; i++) {
14         // Scambia l'elemento arr[i] con l'elemento arr[n-1-i]
15         int temp = arr[i];
16         arr[i] = arr[n - 1 - i];
17         arr[n - 1 - i] = temp;
18     }
19 }
20
21 int main(int argc, char** argv) {
22     int numeri[] = {1, 2, 3, 4, 5, 6, 7};
23     int n = sizeof(numeri) / sizeof(numeri[0]);
24
25     printf("Array originale: ");
26     stampa_array(numeri, n);
27
28     inverti_array(numeri, n);
29
30     printf("Array invertito: ");
31     stampa_array(numeri, n);
32
33     return 0;
34 }
```

### 6.6.4 Copia di un Array

```
1  #include <stdio.h>
2
3  // Funzione che copia gli elementi da un array sorgente a un array
   // destinazione
4  void copia_array(int* sorgente, int* destinazione, int n) {
5      for (int i = 0; i < n; i++) {
6          destinazione[i] = sorgente[i]; // Copia l'i-esimo elemento
7      }
}
```

```
8 }
9
10 int main(int argc, char** argv) {
11     int originale[] = {10, 20, 30, 40, 50};
12     int n = sizeof(originale) / sizeof(originale[0]);
13     int copia[n];
14
15     copia_array(originale, copia, n);
16
17     printf("Array originale: ");
18     for (int i = 0; i < n; i++) {
19         printf("%d ", originale[i]);
20     }
21     printf("\n");
22
23     printf("Array copiato: ");
24     for (int i = 0; i < n; i++) {
25         printf("%d ", copia[i]);
26     }
27     printf("\n");
28
29     return 0;
30 }
```

## 6.7 Array e Funzioni

### 6.7.1 Passaggio di Array a Funzioni

```
1 #include <stdio.h>
2
3 // Funzione che stampa gli elementi di un array di interi
4 // La dimensione deve essere passata come parametro
5 void stampa(int* arr, int n) {
6     for (int i = 0; i < n; i++) {
7         printf("%d ", arr[i]);
8     }
9     printf("\n");
10 }
11
12 // Funzione che calcola la somma degli elementi di un array di interi
13 int somma(int* arr, int n) {
14     int totale = 0;
15     for (int i = 0; i < n; i++) {
16         totale += arr[i];
17     }
18     return totale;
19 }
20
21 // Funzione che raddoppia ogni elemento di un array di interi
22 // Gli array sono passati per riferimento (come puntatori)
23 void raddoppia(int* arr, int n) {
24     for (int i = 0; i < n; i++) {
25         arr[i] *= 2; // Modifica l'elemento nell'array originale
26     }
27 }
28 }
```

```
29 int main(int argc, char** argv) {
30     int numeri[] = {1, 2, 3, 4, 5};
31     int n = sizeof(numeri) / sizeof(numeri[0]);
32
33     printf("Array originale: ");
34     stampa(numeri, n);
35
36     printf("Somma: %d\n", somma(numeri, n));
37
38     raddoppia(numeri, n);
39     printf("Array dopo raddoppio: ");
40     stampa(numeri, n);
41
42     return 0;
43 }
```

### Nota

Quando passi un array a una funzione, in realtà passi un puntatore al primo elemento dell'array (vedi [Capitolo 7](#)). Qualsiasi modifica all'array all'interno della funzione influisce sull'array originale nel chiamante.

## 6.8 Array Multidimensionali

### 6.8.1 Array Bidimensionali (Matrici)

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      // Dichiarazione e inizializzazione
5      int matrice[3][4] = {
6          {1, 2, 3, 4},
7          {5, 6, 7, 8},
8          {9, 10, 11, 12}
9      };
10
11     // Accesso agli elementi
12     printf("Elemento [1][2]: %d\n", matrice[1][2]); // 7
13
14     // Stampa della matrice
15     printf("\nMatrice:\n");
16     for (int i = 0; i < 3; i++) {
17         for (int j = 0; j < 4; j++) {
18             printf("%3d ", matrice[i][j]);
19         }
20         printf("\n");
21     }
22
23     return 0;
24 }
```

### 6.8.2 Operazioni su Matrici

```
1 #include <stdio.h>
2
3 #define RIGHE 3
4 #define COLONNE 3
5
6 // Funzione che stampa una matrice di interi
7 void stampa_matrice(int* mat, int righe) {
8     for (int i = 0; i < righe; i++) {
9         for (int j = 0; j < COLONNE; j++) {
10             printf("%4d", mat[i][j]);
11         }
12         printf("\n");
13     }
14 }
15
16 // Funzione che calcola la somma elemento per elemento di due matrici
    di interi
17 void somma_matrici(int a[][COLONNE], int b[][COLONNE],
18                    int risultato[][COLONNE], int righe) {
19     for (int i = 0; i < righe; i++) {
20         for (int j = 0; j < COLONNE; j++) {
21             risultato[i][j] = a[i][j] + b[i][j]; // Somma gli
                elementi corrispondenti
22         }
23     }
24 }
25
26 // Funzione che calcola la trasposta di una matrice di interi
27 void trasposta(int mat[][COLONNE], int trasposta[][RIGHE],
28                int righe, int colonne) {
29     for (int i = 0; i < righe; i++) {
30         for (int j = 0; j < colonne; j++) {
31             trasposta[j][i] = mat[i][j]; // Scambia righe con colonne
32         }
33     }
34 }
35
36 int main(int argc, char** argv) {
37     int A[RIGHE][COLONNE] = {
38         {1, 2, 3},
39         {4, 5, 6},
40         {7, 8, 9}
41     };
42
43     int B[RIGHE][COLONNE] = {
44         {9, 8, 7},
45         {6, 5, 4},
46         {3, 2, 1}
47     };
48
49     int C[RIGHE][COLONNE];
50     int T[COLONNE][RIGHE];
51
52     printf("Matrice A:\n");
53     stampa_matrice(A, RIGHE);
54
55     printf("\nMatrice B:\n");
```

```
56     stampa_matrice(B, RIGHE);
57
58     somma_matrici(A, B, C, RIGHE);
59     printf("\nSomma A + B:\n");
60     stampa_matrice(C, RIGHE);
61
62     trasposta(A, T, RIGHE, COLONNE);
63     printf("\nTrasposta di A:\n");
64     for (int i = 0; i < COLONNE; i++) {
65         for (int j = 0; j < RIGHE; j++) {
66             printf("%4d", T[i][j]);
67         }
68         printf("\n");
69     }
70
71     return 0;
72 }
```

## 6.9 Algoritmi di Ordinamento

### 6.9.1 Bubble Sort

```
1  #include <stdio.h>
2
3  // Funzione che implementa l'algoritmo Bubble Sort per ordinare un
   // array di interi
4  void bubble_sort(int* arr, int n) {
5      for (int i = 0; i < n - 1; i++) {
6          for (int j = 0; j < n - i - 1; j++) {
7              if (arr[j] > arr[j + 1]) {
8                  // Scambia gli elementi arr[j] e arr[j+1]
9                  int temp = arr[j];
10                 arr[j] = arr[j + 1];
11                 arr[j + 1] = temp;
12             }
13         }
14     }
15 }
16
17 // Funzione che stampa tutti gli elementi di un array di interi
18 void stampa_array(int* arr, int n) {
19     for (int i = 0; i < n; i++) {
20         printf("%d ", arr[i]);
21     }
22     printf("\n");
23 }
24
25 int main(int argc, char** argv) {
26     int numeri[] = {64, 34, 25, 12, 22, 11, 90};
27     int n = sizeof(numeri) / sizeof(numeri[0]);
28
29     printf("Array non ordinato: ");
30     stampa_array(numeri, n);
31
32     bubble_sort(numeri, n);
33 }
```

```
34     printf("Array ordinato: ");
35     stampa_array(numeri, n);
36
37     return 0;
38 }
```

## 6.9.2 Selection Sort

```
1  #include <stdio.h>
2
3  // Funzione che implementa l'algoritmo Selection Sort per ordinare un
   // array di interi
4  void selection_sort(int* arr, int n) {
5      for (int i = 0; i < n - 1; i++) {
6          // Trova l'indice dell'elemento minimo nell'array non ordinato
7          int min_idx = i;
8          for (int j = i + 1; j < n; j++) {
9              if (arr[j] < arr[min_idx]) {
10                 min_idx = j;
11             }
12         }
13         // Scambia l'elemento minimo trovato con il primo elemento non
           // ordinato
14         int temp = arr[min_idx];
15         arr[min_idx] = arr[i];
16         arr[i] = temp;
17     }
18 }
19
20 int main(int argc, char** argv) {
21     int numeri[] = {64, 25, 12, 22, 11};
22     int n = sizeof(numeri) / sizeof(numeri[0]);
23
24     printf("Array non ordinato: ");
25     for (int i = 0; i < n; i++) {
26         printf("%d ", numeri[i]);
27     }
28     printf("\n");
29
30     selection_sort(numeri, n);
31
32     printf("Array ordinato: ");
33     for (int i = 0; i < n; i++) {
34         printf("%d ", numeri[i]);
35     }
36     printf("\n");
37
38     return 0;
39 }
```

## 6.10 Esempi Pratici

### 6.10.1 Statistiche su un Array

```
1 #include <stdio.h>
2
3 // Funzione che calcola e stampa statistiche di un array di interi
4 void statistiche(int* arr, int n) {
5     // Calcola somma, media, massimo e minimo degli elementi
6     int somma = 0;
7     int max = arr[0]; // Inizializza il massimo al primo elemento
8     int min = arr[0]; // Inizializza il minimo al primo elemento
9
10    for (int i = 0; i < n; i++) {
11        somma += arr[i];
12        if (arr[i] > max) max = arr[i];
13        if (arr[i] < min) min = arr[i];
14    }
15
16    float media = (float)somma / n; // Cast a float per divisione
    // decimale
17
18    printf("Statistiche:\n");
19    printf("- Somma: %d\n", somma);
20    printf("- Media: %.2f\n", media);
21    printf("- Massimo: %d\n", max);
22    printf("- Minimo: %d\n", min);
23 }
24
25 int main(int argc, char** argv) {
26     int voti[] = {28, 24, 30, 26, 27, 30, 25};
27     int n = sizeof(voti) / sizeof(voti[0]);
28
29     printf("Voti: ");
30     for (int i = 0; i < n; i++) {
31         printf("%d ", voti[i]);
32     }
33     printf("\n\n");
34
35     statistiche(voti, n);
36
37     return 0;
38 }
```

### 6.10.2 Rimozione Duplicati

```
1 #include <stdio.h>
2
3 // Funzione che rimuove gli elementi duplicati da un array di interi
4 // Restituisce la nuova dimensione dell'array senza duplicati
5 int rimuovi_duplicati(int* arr, int n) {
6     int nuova_dimensioe = 0;
7
8     for (int i = 0; i < n; i++) {
9         int duplicato = 0;
10        // Controlla se l'elemento arr[i] e' gia' presente nell'array
        // risultante
11        for (int j = 0; j < nuova_dimensioe; j++) {
12            if (arr[i] == arr[j]) {
13                duplicato = 1;
```

```

14         break;
15     }
16 }
17 // Se l'elemento non e' duplicato, aggiungilo all'array
18 risultante
19 if (!duplicato) {
20     arr[nuova_dimensione] = arr[i];
21     nuova_dimensione++;
22 }
23
24 return nuova_dimensione; // Restituisce la dimensione dell'array
25 senza duplicati
26 }
27
28 int main(int argc, char** argv) {
29     int numeri[] = {1, 2, 3, 2, 4, 1, 5, 3, 6};
30     int n = sizeof(numeri) / sizeof(numeri[0]);
31
32     printf("Array originale: ");
33     for (int i = 0; i < n; i++) {
34         printf("%d ", numeri[i]);
35     }
36     printf("\n");
37
38     n = rimuovi_duplicati(numeri, n);
39
40     printf("Array senza duplicati: ");
41     for (int i = 0; i < n; i++) {
42         printf("%d ", numeri[i]);
43     }
44     printf("\n");
45
46     return 0;
47 }

```

## 6.11 Esercizi

### 6.11.1 Livello Base

1. Scrivi un programma che legge 10 numeri e stampa quanti sono pari e quanti dispari.
2. Crea un programma che trova la seconda cifra più grande in un array.
3. Implementa un programma che conta le occorrenze di un numero in un array.
4. Scrivi un programma che verifica se un array è palindromo.

### 6.11.2 Livello Intermedio

1. Implementa la ricerca binaria su un array ordinato.
2. Crea un programma che ruota gli elementi di un array di K posizioni.
3. Scrivi un programma che fonde (merge) due array ordinati in un unico array ordinato.

4. Implementa un programma che trova tutti i sottoinsiemi di somma uguale a un valore dato.

### 6.11.3 Livello Avanzato

1. Implementa l'algoritmo Quick Sort ricorsivo.
2. Crea un programma per la moltiplicazione di due matrici.
3. Scrivi un programma che trova la sotto-sequenza contigua con somma massima (algoritmo di Kadane).
4. Implementa un programma che risolve il problema dello zaino (knapsack) 0/1 con programmazione dinamica.

# Capitolo 7

## Puntatori

### 7.1 Introduzione ai Puntatori

I puntatori sono una delle caratteristiche più potenti e complesse del linguaggio C. Un puntatore è una variabile speciale che contiene l'indirizzo di memoria di un'altra variabile, permettendo l'accesso e la manipolazione indiretta dei dati.

#### 7.1.1 Obiettivi di Apprendimento

Al termine di questo capitolo sarai in grado di:

- Comprendere il concetto di indirizzo di memoria e puntatore
- Dichiarare e inizializzare correttamente i puntatori
- Utilizzare gli operatori `&` (indirizzo) e `*` (dereferenziazione) del linguaggio C
- Applicare l'aritmetica dei puntatori per navigare in memoria
- Comprendere la relazione tra array e puntatori (vedi [Capitolo 6](#))
- Passare puntatori alle funzioni per modificare variabili (vedi [Capitolo 5](#))
- Allocare e deallocare memoria dinamicamente
- Riconoscere ed evitare gli errori comuni con i puntatori

#### 7.1.2 Perché Usare i Puntatori?

- Manipolazione diretta della memoria
- Passaggio efficiente di dati alle funzioni
- Allocazione dinamica della memoria
- Implementazione di strutture dati complesse
- Accesso efficiente agli array

**Attenzione**

I puntatori sono potenti ma possono causare errori gravi se usati incorrettamente. È fondamentale comprenderli bene prima di usarli.

## 7.2 Concetti Fondamentali

### 7.2.1 Indirizzi di Memoria

Ogni variabile è memorizzata in una specifica posizione di memoria, identificata da un indirizzo.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int x = 10; // Variabile intera
5
6     printf("Valore di x: %d\n", x);
7     printf("Indirizzo di x: %p\n", (void*)&x); // Stampa l'indirizzo
8         di memoria della variabile x
9
10    return 0;
11 }
```

Output (esempio):

Valore di x: 10

Indirizzo di x: 0x7ffeeb3c4a4c

### 7.2.2 Operatori dei Puntatori

- & (operatore di indirizzo del linguaggio C): restituisce l'indirizzo di memoria di una variabile
- \* (operatore di dereferenziazione del linguaggio C): accede al valore contenuto all'indirizzo puntato da un puntatore

## 7.3 Dichiarazione e Inizializzazione

### 7.3.1 Sintassi

```
1 tipo *nome_puntatore;
```

### 7.3.2 Esempi

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int x = 10; // Variabile intera
5     int *ptr; // Dichiarazione di un puntatore a variabile di tipo
6         int
```

```
7     ptr = &x; // Il puntatore ptr contiene l'indirizzo di memoria
           della variabile x
8
9     printf("Valore di x: %d\n", x);
10    printf("Indirizzo di x: %p\n", (void*)&x);
11    printf("Valore di ptr: %p\n", (void*)ptr); // Stampa l'indirizzo
           contenuto nel puntatore
12    printf("Valore puntato da ptr: %d\n", *ptr); // Dereferenzia il
           puntatore per ottenere il valore
13
14    return 0;
15 }
```

### 7.3.3 Inizializzazione Diretta

```
1 int x = 42; // Variabile intera
2 int *ptr = &x; // Dichiarazione e inizializzazione del puntatore all'
           indirizzo di x
3
4 // Oppure
5 int y = 100; // Variabile intera
6 int *p1 = &y, *p2 = &y; // Due puntatori distinti che puntano alla
           stessa variabile y
```

## 7.4 Dereferenziazione

La dereferenziazione, tramite l'operatore \*, permette di accedere o modificare il valore contenuto all'indirizzo di memoria puntato da un puntatore.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int x = 10; // Variabile intera
5     int *ptr = &x; // Puntatore che punta alla variabile x
6
7     printf("Valore originale: %d\n", x);
8
9     // Modifica del valore tramite dereferenziazione del puntatore
10    *ptr = 20; // Modifica indirettamente la variabile x
11
12    printf("Valore modificato: %d\n", x); // 20
13    printf("Valore tramite puntatore: %d\n", *ptr); // 20,
           dereferenzia il puntatore
14
15    return 0;
16 }
```

## 7.5 Puntatori e Funzioni

### 7.5.1 Passaggio per Riferimento

I puntatori permettono di modificare le variabili passate alle funzioni.

```
1 #include <stdio.h>
2
3 // Funzione con passaggio per valore (non modifica la variabile
  originale)
4 void incrementa_valore(int x) {
5     x = x + 1; // Modifica solo la copia locale del parametro
6 }
7
8 // Funzione con passaggio per riferimento tramite puntatore (modifica
  la variabile originale)
9 void incrementa_riferimento(int *x) {
10     *x = *x + 1; // Dereferenzia il puntatore e modifica il valore
    puntato
11 }
12
13 int main(int argc, char** argv) {
14     int numero = 10;
15
16     printf("Valore iniziale: %d\n", numero);
17
18     incrementa_valore(numero); // Passa il valore per copia
19     printf("Dopo incrementa_valore: %d\n", numero); // Ancora 10 (non
    modificato)
20
21     incrementa_riferimento(&numero); // Passa l'indirizzo della
    variabile
22     printf("Dopo incrementa_riferimento: %d\n", numero); // 11 (
    modificato)
23
24     return 0;
25 }
```

### 7.5.2 Scambio di Valori

```
1 #include <stdio.h>
2
3 // Funzione che scambia i valori di due variabili intere tramite
  puntatori
4 void scambia(int *a, int *b) {
5     int temp = *a; // Salva il valore puntato da a
6     *a = *b; // Assegna il valore puntato da b alla variabile puntata
    da a
7     *b = temp; // Assegna il valore salvato alla variabile puntata da
    b
8 }
9
10 int main(int argc, char** argv) {
11     int x = 5, y = 10;
12
13     printf("Prima: x = %d, y = %d\n", x, y);
14     scambia(&x, &y); // Passa gli indirizzi delle variabili x e y
    alla funzione
15     printf("Dopo: x = %d, y = %d\n", x, y); // I valori sono stati
    scambiati
16
17     return 0;
18 }
```

```
18 }
```

### 7.5.3 Ritorno di Valori Multipli

```
1 #include <stdio.h>
2
3 // Funzione che calcola quoziente e resto e li restituisce tramite
  // puntatori
4 void dividi(int dividendo, int divisore,
5             int *quoziente, int *resto) {
6     *quoziente = dividendo / divisore; // Assegna il quoziente alla
      // variabile puntata
7     *resto = dividendo % divisore; // Assegna il resto alla variabile
      // puntata
8 }
9
10 int main(int argc, char** argv) {
11     int q, r; // Variabili per quoziente e resto
12
13     dividi(17, 5, &q, &r); // Passa gli indirizzi delle variabili q e
      // r alla funzione
14
15     printf("17 / 5 = %d con resto %d\n", q, r);
16
17     return 0;
18 }
```

## 7.6 Puntatori e Array

### 7.6.1 Relazione tra Array e Puntatori

Il nome di un array in C è un puntatore costante al primo elemento dell'array stesso (vedi [Capitolo 6](#)).

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int arr[5] = {10, 20, 30, 40, 50}; // Array di 5 interi
5     int *ptr = arr; // Il puntatore ptr punta al primo elemento (
      // equivalente a &arr[0])
6
7     printf("arr[0] = %d\n", arr[0]); // 10, accesso con notazione
      // array
8     printf("*arr = %d\n", *arr); // 10, dereferenzia il
      // puntatore al primo elemento
9     printf("*ptr = %d\n", *ptr); // 10, dereferenzia il
      // puntatore ptr
10
11     // Accesso agli elementi con diverse notazioni
12     printf("arr[2] = %d\n", arr[2]); // 30, accesso con indice
13     printf("*(arr + 2) = %d\n", *(arr + 2)); // 30, aritmetica dei
      // puntatori e dereferenziazione
14     printf("*(ptr + 2) = %d\n", *(ptr + 2)); // 30, aritmetica dei
      // puntatori e dereferenziazione
```

```
15
16     return 0;
17 }
```

## 7.6.2 Aritmetica dei Puntatori

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      int arr[5] = {10, 20, 30, 40, 50}; // Array di 5 interi
5      int *ptr = arr; // Puntatore che punta al primo elemento dell'
        array
6
7      printf("Attraversamento con aritmetica dei puntatori:\n");
8      for (int i = 0; i < 5; i++) {
9          printf("*(ptr + %d) = %d\n", i, *(ptr + i)); // Dereferenzia
        puntatore + offset
10     }
11
12     printf("\nUsando incremento del puntatore:\n");
13     ptr = arr; // Reset del puntatore al primo elemento
14     for (int i = 0; i < 5; i++) {
15         printf("*ptr = %d\n", *ptr); // Dereferenzia il puntatore
        corrente
16         ptr++; // Incrementa il puntatore per avanzare al prossimo
        elemento
17     }
18
19     return 0;
20 }
```

## 7.6.3 Passaggio di Array a Funzioni con Puntatori

```
1  #include <stdio.h>
2
3  // Funzione che stampa un array usando notazione puntatore
4  // Queste dichiarazioni di parametro sono equivalenti: int *arr e int
    arr[]
5  void stampa1(int* arr, int n) {
6      for (int i = 0; i < n; i++) {
7          printf("%d ", arr[i]); // Accesso con notazione array
8      }
9      printf("\n");
10 }
11
12 // Funzione che stampa un array usando aritmetica dei puntatori
13 void stampa2(int* arr, int n) {
14     for (int i = 0; i < n; i++) {
15         printf("%d ", *(arr + i)); // Accesso con aritmetica dei
        puntatori
16     }
17     printf("\n");
18 }
19
20 // Funzione che modifica gli elementi di un array tramite puntatore
```

```

21 void modifica(int* arr, int n) {
22     for (int i = 0; i < n; i++) {
23         arr[i] *= 2; // Modifica l'elemento nell'array originale
24     }
25 }
26
27 int main(int argc, char** argv) {
28     int numeri[] = {1, 2, 3, 4, 5};
29     int n = sizeof(numeri) / sizeof(numeri[0]);
30
31     stampa1(numeri, n);
32     stampa2(numeri, n);
33
34     modifica(numeri, n);
35     printf("Dopo modifica: ");
36     stampa1(numeri, n);
37
38     return 0;
39 }

```

## 7.7 Puntatori a Puntatori

Un puntatore può puntare a un altro puntatore, creando una catena di indirizioni.

```

1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      int x = 100; // Variabile intera
5      int *ptr1 = &x; // Puntatore a variabile int che punta a x
6      int **ptr2 = &ptr1; // Puntatore a puntatore a int che punta a
                          ptr1
7
8      printf("Valore di x: %d\n", x);
9      printf("Valore tramite *ptr1: %d\n", *ptr1); // Dereferenzia ptr1
10     printf("Valore tramite **ptr2: %d\n", **ptr2); // Doppia
                          dereferenziazione: prima ptr2 poi ptr1
11
12     // Modifica tramite puntatore a puntatore
13     **ptr2 = 200; // Dereferenzia due volte per modificare x
14     printf("Nuovo valore di x: %d\n", x);
15
16     return 0;
17 }

```

## 7.8 Puntatori void

Un puntatore di tipo `void*` può puntare a qualsiasi tipo di dato, ma richiede un cast esplicito prima della dereferenziazione.

```

1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      int x = 10; // Variabile intera
5      float y = 3.14; // Variabile float

```

```

6   char c = 'A'; // Variabile char
7
8   void *ptr; // Puntatore generico void
9
10  // Il puntatore void punta a una variabile int
11  ptr = &x;
12  printf("Intero: %d\n", *(int*)ptr); // Cast a int* prima di
    dereferenziare
13
14  // Il puntatore void punta a una variabile float
15  ptr = &y;
16  printf("Float: %.2f\n", *(float*)ptr); // Cast a float* prima di
    dereferenziare
17
18  // Il puntatore void punta a una variabile char
19  ptr = &c;
20  printf("Char: %c\n", *(char*)ptr); // Cast a char* prima di
    dereferenziare
21
22  return 0;
23 }

```

### Attenzione

Prima di dereferenziare un puntatore di tipo void, devi fare il cast esplicito al tipo appropriato del dato puntato, altrimenti il compilatore non sa come interpretare i dati in memoria.

## 7.9 Puntatori Costanti

### 7.9.1 Tipi di Costanti

```

1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      int x = 10, y = 20;
5
6      // 1. Puntatore a costante
7      const int *ptr1 = &x;
8      // *ptr1 = 20; // ERRORE: non puoi modificare il valore
9      ptr1 = &y;     // OK: puoi cambiare dove punta
10
11     // 2. Puntatore costante
12     int *const ptr2 = &x;
13     *ptr2 = 30;     // OK: puoi modificare il valore
14     // ptr2 = &y;   // ERRORE: non puoi cambiare dove punta
15
16     // 3. Puntatore costante a costante
17     const int *const ptr3 = &x;
18     // *ptr3 = 40; // ERRORE: non puoi modificare il valore
19     // ptr3 = &y;   // ERRORE: non puoi cambiare dove punta
20
21     return 0;
22 }

```

## 7.10 Puntatore NULL

Il puntatore NULL è un puntatore che non punta a nulla.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     int *ptr = NULL; // Inizializzazione sicura
6
7     if (ptr == NULL) {
8         printf("Il puntatore e' NULL\n");
9     }
10
11     // Sempre verificare prima di dereferenziare
12     if (ptr != NULL) {
13         printf("Valore: %d\n", *ptr);
14     } else {
15         printf("Impossibile dereferenziare: puntatore NULL\n");
16     }
17
18     return 0;
19 }
```

### Nota

È buona pratica inizializzare sempre i puntatori a NULL se non vengono immediatamente assegnati a un indirizzo valido.

## 7.11 Allocazione Dinamica della Memoria

### 7.11.1 malloc

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     int n;
6     printf("Quanti numeri? ");
7     scanf("%d", &n);
8
9     // Alloca memoria per n interi
10    int *arr = (int*)malloc(n * sizeof(int));
11
12    if (arr == NULL) {
13        printf("Errore di allocazione memoria\n");
14        return 1;
15    }
16
17    // Usa l'array
18    for (int i = 0; i < n; i++) {
19        arr[i] = i * 2;
20    }
21
22    printf("Array: ");
```

```
23     for (int i = 0; i < n; i++) {
24         printf("%d ", arr[i]);
25     }
26     printf("\n");
27
28     // Libera la memoria
29     free(arr);
30
31     return 0;
32 }
```

### 7.11.2 calloc

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv) {
5      int n = 5;
6
7      // calloc inizializza a zero
8      int *arr = (int*)calloc(n, sizeof(int));
9
10     if (arr == NULL) {
11         printf("Errore di allocazione\n");
12         return 1;
13     }
14
15     printf("Array inizializzato da calloc: ");
16     for (int i = 0; i < n; i++) {
17         printf("%d ", arr[i]); // Tutti zero
18     }
19     printf("\n");
20
21     free(arr);
22     return 0;
23 }
```

### 7.11.3 realloc

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv) {
5      int *arr = (int*)malloc(3 * sizeof(int));
6
7      arr[0] = 10;
8      arr[1] = 20;
9      arr[2] = 30;
10
11     printf("Array originale: ");
12     for (int i = 0; i < 3; i++) {
13         printf("%d ", arr[i]);
14     }
15     printf("\n");
16 }
```

```
17 // Ridimensiona l'array
18 arr = (int*)realloc(arr, 5 * sizeof(int));
19
20 arr[3] = 40;
21 arr[4] = 50;
22
23 printf("Array ridimensionato: ");
24 for (int i = 0; i < 5; i++) {
25     printf("%d ", arr[i]);
26 }
27 printf("\n");
28
29 free(arr);
30 return 0;
31 }
```

## 7.12 Errori Comuni con i Puntatori

### 7.12.1 Puntatore Non Inizializzato

```
1 // SBAGLIATO
2 int *ptr;
3 *ptr = 10; // Comportamento indefinito!
4
5 // CORRETTO
6 int x;
7 int *ptr = &x;
8 *ptr = 10; // OK
```

### 7.12.2 Dereferenziazione di NULL

```
1 // SBAGLIATO
2 int *ptr = NULL;
3 *ptr = 10; // Crash!
4
5 // CORRETTO
6 int *ptr = NULL;
7 if (ptr != NULL) {
8     *ptr = 10;
9 }
```

### 7.12.3 Memory Leak

```
1 // SBAGLIATO
2 void funzione() {
3     int *ptr = (int*)malloc(100 * sizeof(int));
4     // ... uso del puntatore ...
5     // Manca free(ptr)!
6 }
7
8 // CORRETTO
9 void funzione() {
```

```
10     int *ptr = (int*)malloc(100 * sizeof(int));
11     if (ptr == NULL) return;
12
13     // ... uso del puntatore ...
14
15     free(ptr); // Libera la memoria
16 }
```

### 7.12.4 Dangling Pointer

```
1 // SBAGLIATO
2 int *ptr = (int*)malloc(sizeof(int));
3 *ptr = 10;
4 free(ptr);
5 *ptr = 20; // Errore: ptr punta a memoria liberata!
6
7 // CORRETTO
8 int *ptr = (int*)malloc(sizeof(int));
9 *ptr = 10;
10 free(ptr);
11 ptr = NULL; // Imposta a NULL dopo free
```

## 7.13 Esempi Pratici

### 7.13.1 Lista Dinamica di Numeri

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     int capacita = 2;
6     int dimensione = 0;
7     int *arr = (int*)malloc(capacita * sizeof(int));
8
9     if (arr == NULL) {
10         printf("Errore di allocazione\n");
11         return 1;
12     }
13
14     int numero;
15     printf("Inserisci numeri (0 per terminare):\n");
16
17     while (1) {
18         scanf("%d", &numero);
19         if (numero == 0) break;
20
21         // Espandi l'array se necessario
22         if (dimensione == capacita) {
23             capacita *= 2;
24             arr = (int*)realloc(arr, capacita * sizeof(int));
25             if (arr == NULL) {
26                 printf("Errore di riallocazione\n");
27                 return 1;
28             }
29         }
30     }
31 }
```

```
29         printf("Array espanso a capacita %d\n", capacita);
30     }
31
32     arr[dimensione++] = numero;
33 }
34
35 printf("\nNumeri inseriti:\n");
36 for (int i = 0; i < dimensione; i++) {
37     printf("%d ", arr[i]);
38 }
39 printf("\n");
40
41 free(arr);
42 return 0;
43 }
```

## 7.14 Esercizi

### 7.14.1 Livello Base

1. Scrivi una funzione che usa i puntatori per scambiare tre valori ciclicamente ( $a \rightarrow b$ ,  $b \rightarrow c$ ,  $c \rightarrow a$ ).
2. Crea una funzione che inverte un array usando i puntatori.
3. Implementa una funzione che trova il massimo e il minimo di un array restituendo entrambi tramite puntatori.
4. Scrivi un programma che alloca dinamicamente un array e lo riempie con i quadrati dei numeri da 1 a N.

### 7.14.2 Livello Intermedio

1. Implementa una funzione che concatena due array allocati dinamicamente in un nuovo array.
2. Crea una funzione che rimuove gli elementi duplicati da un array allocato dinamicamente.
3. Scrivi un programma che implementa una matrice dinamica (array 2D allocato dinamicamente).
4. Implementa una funzione di ordinamento che usa solo aritmetica dei puntatori (senza indici).

### 7.14.3 Livello Avanzato

1. Implementa una lista collegata semplice con operazioni di inserimento, ricerca e cancellazione.
2. Crea un programma che implementa uno stack dinamico usando puntatori.
3. Scrivi un programma che gestisce una matrice sparsa usando allocazione dinamica.
4. Implementa un albero binario di ricerca con allocazione dinamica dei nodi.

# Capitolo 8

## Stringhe

### 8.1 Introduzione alle Stringhe

In C, le stringhe sono array di caratteri terminati da un carattere speciale: `'\0'` (carattere nullo o null terminator).

#### 8.1.1 Obiettivi di Apprendimento

Alla fine di questo capitolo sarai in grado di:

- Comprendere come le stringhe sono rappresentate in C come array di caratteri
- Utilizzare il terminatore null `'\0'` correttamente
- Dichiarare e inizializzare stringhe in vari modi
- Leggere e stampare stringhe con diverse tecniche di I/O
- Utilizzare le principali funzioni della libreria `string.h`
- Manipolare stringhe (conversione, inversione, concatenazione)
- Evitare buffer overflow e altri errori comuni con le stringhe
- Creare array di stringhe e gestire strutture dati complesse
- Convertire tra stringhe e numeri in entrambe le direzioni

#### Nota

A differenza di altri linguaggi, C non ha un tipo di dato "stringa" nativo. Le stringhe sono gestite come array di `char`.

#### 8.1.2 Caratteristiche delle Stringhe

- Array di caratteri con terminatore null
- Il terminatore `'\0'` marca la fine della stringa
- La dimensione dell'array deve includere spazio per il terminatore
- Possono essere manipolate come array o tramite puntatori

## 8.2 Dichiarazione e Inizializzazione

### 8.2.1 Modi di Dichiarare Stringhe

```
1 #include <stdio.h>
2
3 int main() {
4     // Metodo 1: Array di caratteri con inizializzazione
5     char str1[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
6
7     // Metodo 2: Inizializzazione con letterale stringa
8     char str2[] = "Hello"; // Dimensione automatica (6 caratteri)
9
10    // Metodo 3: Specifica dimensione esplicita
11    char str3[20] = "Hello"; // str3 ha 20 caratteri, usa solo 6
12
13    // Metodo 4: Puntatore a stringa letterale (costante)
14    char *str4 = "Hello"; // Non modificabile!
15
16    printf("%s\n", str1);
17    printf("%s\n", str2);
18    printf("%s\n", str3);
19    printf("%s\n", str4);
20
21    return 0;
22 }
```

#### Attenzione

Le stringhe letterali (come "Hello") sono memorizzate in una zona di memoria di sola lettura. Non tentare di modificarle!

### 8.2.2 Differenza tra Array e Puntatore

```
1 #include <stdio.h>
2
3 int main() {
4     // Array di caratteri (modificabile)
5     char str1[] = "Hello";
6     str1[0] = 'J'; // OK: modifica in "Jello"
7
8     // Puntatore a stringa letterale (non modificabile)
9     char *str2 = "Hello";
10    // str2[0] = 'J'; // ERRORE: crash a runtime!
11
12    printf("%s\n", str1);
13    printf("%s\n", str2);
14
15    return 0;
16 }
```

## 8.3 Input e Output di Stringhe

### 8.3.1 Funzioni di I/O

```
1 #include <stdio.h>
2
3 int main() {
4     char nome[50];
5
6     // Metodo 1: scanf (legge fino allo spazio)
7     printf("Inserisci nome: ");
8     scanf("%s", nome); // Nota: nome, non &nome!
9     printf("Nome: %s\n", nome);
10
11    // Pulisci il buffer
12    while (getchar() != '\n');
13
14    // Metodo 2: gets (deprecato, non usare!)
15    // gets(nome); // PERICOLOSO: non controlla i limiti!
16
17    // Metodo 3: fgets (sicuro, legge anche spazi)
18    printf("Inserisci nome completo: ");
19    fgets(nome, sizeof(nome), stdin);
20    printf("Nome completo: %s", nome);
21
22    return 0;
23 }
```

#### Attenzione

`scanf("%s", str)` si ferma al primo spazio. Usa `fgets()` per leggere stringhe con spazi.

### 8.3.2 Carattere per Carattere

```
1 #include <stdio.h>
2
3 int main() {
4     char str[100];
5     char ch;
6     int i = 0;
7
8     printf("Inserisci una stringa (max 99 caratteri):\n");
9
10    // Lettura carattere per carattere
11    while ((ch = getchar()) != '\n' && i < 99) {
12        str[i++] = ch;
13    }
14    str[i] = '\0'; // Aggiungi terminatore
15
16    printf("Hai inserito: %s\n", str);
17
18    return 0;
19 }
```

## 8.4 Funzioni Standard per Stringhe

La libreria `string.h` fornisce molte funzioni utili.

### 8.4.1 `strlen` - Lunghezza della Stringa

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char str[] = "Hello World";
6
7     int lunghezza = strlen(str);
8     printf("Lunghezza: %d\n", lunghezza); // 11
9
10    // Implementazione manuale
11    int len = 0;
12    while (str[len] != '\0') {
13        len++;
14    }
15    printf("Lunghezza (manuale): %d\n", len);
16
17    return 0;
18 }
```

### 8.4.2 `strcpy` - Copia di Stringhe

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char sorgente[] = "Hello";
6     char destinazione[20];
7
8     // Copia sorgente in destinazione
9     strcpy(destinazione, sorgente);
10    printf("Destinazione: %s\n", destinazione);
11
12    // Versione sicura con limite
13    strncpy(destinazione, "World", sizeof(destinazione) - 1);
14    destinazione[sizeof(destinazione) - 1] = '\0';
15    printf("Destinazione: %s\n", destinazione);
16
17    return 0;
18 }
```

### 8.4.3 `strcat` - Concatenazione

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char str1[50] = "Hello ";
```

```
6     char str2[] = "World";
7
8     // Concatena str2 a str1
9     strcat(str1, str2);
10    printf("Risultato: %s\n", str1); // "Hello World"
11
12    // Versione sicura con limite
13    char str3[20] = "C ";
14    strncat(str3, "Programming", sizeof(str3) - strlen(str3) - 1);
15    printf("Risultato: %s\n", str3);
16
17    return 0;
18 }
```

#### 8.4.4 strcmp - Confronto di Stringhe

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char str1[] = "apple";
6     char str2[] = "banana";
7     char str3[] = "apple";
8
9     // strcmp restituisce:
10    // < 0 se str1 < str2
11    // 0 se str1 == str2
12    // > 0 se str1 > str2
13
14    if (strcmp(str1, str2) < 0) {
15        printf("%s viene prima di %s\n", str1, str2);
16    }
17
18    if (strcmp(str1, str3) == 0) {
19        printf("%s e' uguale a %s\n", str1, str3);
20    }
21
22    // Confronto case-insensitive (non standard ovunque)
23    #ifdef _WIN32
24    if (stricmp(str1, "APPLE") == 0) {
25        printf("Uguali ignorando maiuscole/minuscole\n");
26    }
27    #else
28    if (strcasecmp(str1, "APPLE") == 0) {
29        printf("Uguali ignorando maiuscole/minuscole\n");
30    }
31    #endif
32
33    return 0;
34 }
```

#### 8.4.5 Altre Funzioni Utili

```
1 #include <stdio.h>
2 #include <string.h>
```

```
3 #include <ctype.h>
4
5 int main() {
6     char str[] = "Hello World 123";
7
8     // strchr: trova un carattere
9     char *pos = strchr(str, 'o');
10    if (pos != NULL) {
11        printf("Prima 'o' trovata alla posizione %ld\n",
12              pos - str);
13    }
14
15    // strstr: trova una sottostringa
16    pos = strstr(str, "World");
17    if (pos != NULL) {
18        printf("'World' trovato: %s\n", pos);
19    }
20
21    // strrchr: trova l'ultima occorrenza
22    pos = strrchr(str, 'o');
23    if (pos != NULL) {
24        printf("Ultima 'o' alla posizione %ld\n", pos - str);
25    }
26
27    return 0;
28 }
```

## 8.5 Manipolazione di Stringhe

### 8.5.1 Conversione Maiuscolo/Minuscolo

```
1 #include <stdio.h>
2 #include <ctype.h>
3
4 void maiuscolo(char *str) {
5     for (int i = 0; str[i] != '\0'; i++) {
6         str[i] = toupper(str[i]);
7     }
8 }
9
10 void minuscolo(char *str) {
11     for (int i = 0; str[i] != '\0'; i++) {
12         str[i] = tolower(str[i]);
13     }
14 }
15
16 int main() {
17     char str1[] = "Hello World";
18     char str2[] = "HELLO WORLD";
19
20     maiuscolo(str1);
21     printf("Maiuscolo: %s\n", str1);
22
23     minuscolo(str2);
24     printf("Minuscolo: %s\n", str2);
25 }
```

```
26     return 0;
27 }
```

### 8.5.2 Inversione di una Stringa

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void inverti(char *str) {
5      int n = strlen(str);
6      for (int i = 0; i < n / 2; i++) {
7          char temp = str[i];
8          str[i] = str[n - 1 - i];
9          str[n - 1 - i] = temp;
10     }
11 }
12
13 int main() {
14     char str[] = "Hello";
15
16     printf("Originale: %s\n", str);
17     inverti(str);
18     printf("Invertita: %s\n", str);
19
20     return 0;
21 }
```

### 8.5.3 Rimozione degli Spazi

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  void rimuovi_spazi(char *str) {
5      int i = 0, j = 0;
6
7      while (str[i] != '\0') {
8          if (!isspace(str[i])) {
9              str[j++] = str[i];
10         }
11         i++;
12     }
13     str[j] = '\0';
14 }
15
16 int main() {
17     char str[] = "H e l l o   W o r l d";
18
19     printf("Originale: [%s]\n", str);
20     rimuovi_spazi(str);
21     printf("Senza spazi: [%s]\n", str);
22
23     return 0;
24 }
```

### 8.5.4 Conta Parole

```
1 #include <stdio.h>
2 #include <ctype.h>
3
4 int conta_parole(char *str) {
5     int conta = 0;
6     int in_parola = 0;
7
8     for (int i = 0; str[i] != '\0'; i++) {
9         if (isspace(str[i])) {
10             in_parola = 0;
11         } else if (!in_parola) {
12             in_parola = 1;
13             conta++;
14         }
15     }
16
17     return conta;
18 }
19
20 int main() {
21     char str[] = "Questa e' una frase di esempio";
22
23     int parole = conta_parole(str);
24     printf("La stringa contiene %d parole\n", parole);
25
26     return 0;
27 }
```

## 8.6 Array di Stringhe

### 8.6.1 Array Bidimensionale di char

```
1 #include <stdio.h>
2
3 int main() {
4     // Array di 5 stringhe, ognuna max 20 caratteri
5     char nomi[5][20] = {
6         "Mario",
7         "Lucia",
8         "Giovanni",
9         "Anna",
10        "Francesco"
11    };
12
13    printf("Lista nomi:\n");
14    for (int i = 0; i < 5; i++) {
15        printf("%d. %s\n", i + 1, nomi[i]);
16    }
17
18    return 0;
19 }
```

### 8.6.2 Array di Puntatori a Stringhe

```
1 #include <stdio.h>
2
3 int main() {
4     // Array di puntatori a stringhe
5     char *giorni[] = {
6         "Lunedì",
7         "Martedì",
8         "Mercoledì",
9         "Giovedì",
10        "Venerdì",
11        "Sabato",
12        "Domenica"
13    };
14
15    printf("Giorni della settimana:\n");
16    for (int i = 0; i < 7; i++) {
17        printf("%s\n", giorni[i]);
18    }
19
20    return 0;
21 }
```

## 8.7 Conversioni tra Stringhe e Numeri

### 8.7.1 Da Stringa a Numero

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     char str_int[] = "12345";
6     char str_float[] = "3.14159";
7     char str_long[] = "9876543210";
8
9     // Conversione a int
10    int num = atoi(str_int);
11    printf("Intero: %d\n", num);
12
13    // Conversione a float
14    float fnum = atof(str_float);
15    printf("Float: %.2f\n", fnum);
16
17    // Conversione a long
18    long lnum = atol(str_long);
19    printf("Long: %ld\n", lnum);
20
21    // Conversione con strtol (piu' robusta)
22    char *endptr;
23    long val = strtol(" 123abc", &endptr, 10);
24    printf("Valore: %ld, Resto: %s\n", val, endptr);
25
26    return 0;
27 }
```

## 8.7.2 Da Numero a Stringa

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int num = 12345;
6     float fnum = 3.14159;
7
8     char buffer[50];
9
10    // sprintf: stampa su stringa
11    sprintf(buffer, "%d", num);
12    printf("Intero come stringa: %s\n", buffer);
13
14    sprintf(buffer, "%.2f", fnum);
15    printf("Float come stringa: %s\n", buffer);
16
17    // Formato complesso
18    sprintf(buffer, "x=%d, y=%.1f", num, fnum);
19    printf("Formato: %s\n", buffer);
20
21    return 0;
22 }
```

## 8.8 Tokenizzazione di Stringhe

### 8.8.1 Uso di strtok

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char str[] = "apple,banana,cherry,date";
6     char *token;
7
8     printf("Parole separate da virgola:\n");
9
10    // Primo token
11    token = strtok(str, ",");
12
13    // Token successivi
14    while (token != NULL) {
15        printf("- %s\n", token);
16        token = strtok(NULL, ",");
17    }
18
19    // Esempio con spazi multipli
20    char frase[] = "Questa e' una frase";
21    printf("\nParole nella frase:\n");
22
23    token = strtok(frase, " ");
24    while (token != NULL) {
25        printf("- %s\n", token);
26        token = strtok(NULL, " ");
27    }
```

```
27     }
28
29     return 0;
30 }
```

### Attenzione

`strtok()` modifica la stringa originale inserendo caratteri null. Se hai bisogno della stringa originale, fai prima una copia.

## 8.9 Validazione di Stringhe

### 8.9.1 Verifica se è un Numero

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int is_numero(char *str) {
5      if (*str == '-' || *str == '+') {
6          str++; // Salta il segno
7      }
8
9      if (*str == '\0') {
10         return 0; // Stringa vuota o solo segno
11     }
12
13     while (*str != '\0') {
14         if (!isdigit(*str)) {
15             return 0;
16         }
17         str++;
18     }
19
20     return 1;
21 }
22
23 int main() {
24     char str1[] = "12345";
25     char str2[] = "-678";
26     char str3[] = "123abc";
27     char str4[] = "";
28
29     printf("%s: %s\n", str1, is_numero(str1) ? "Numero" : "Non numero");
30     printf("%s: %s\n", str2, is_numero(str2) ? "Numero" : "Non numero");
31     printf("%s: %s\n", str3, is_numero(str3) ? "Numero" : "Non numero");
32     printf("%s: %s\n", str4, is_numero(str4) ? "Numero" : "Non numero");
33
34     return 0;
35 }
```

### 8.9.2 Verifica Palindromo

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 int is_palindromo(char *str) {
6     int sinistra = 0;
7     int destra = strlen(str) - 1;
8
9     while (sinistra < destra) {
10        // Salta caratteri non alfanumerici
11        while (sinistra < destra && !isalnum(str[sinistra])) {
12            sinistra++;
13        }
14        while (sinistra < destra && !isalnum(str[destra])) {
15            destra--;
16        }
17
18        // Confronta ignorando maiuscole/minuscole
19        if (tolower(str[sinistra]) != tolower(str[destra])) {
20            return 0;
21        }
22
23        sinistra++;
24        destra--;
25    }
26
27    return 1;
28 }
29
30 int main() {
31     char str1[] = "radar";
32     char str2[] = "hello";
33     char str3[] = "A man, a plan, a canal: Panama";
34
35     printf("%s: %s\n", str1,
36           is_palindromo(str1) ? "Palindromo" : "Non palindromo");
37     printf("%s: %s\n", str2,
38           is_palindromo(str2) ? "Palindromo" : "Non palindromo");
39     printf("%s: %s\n", str3,
40           is_palindromo(str3) ? "Palindromo" : "Non palindromo");
41
42     return 0;
43 }
```

## 8.10 Esempi Pratici

### 8.10.1 Sistema di Login

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX_USERNAME 20
5 #define MAX_PASSWORD 20
```

```
6
7 int verifica_login(char *username, char *password) {
8     // Username e password corretti (esempio)
9     char user_corretto[] = "admin";
10    char pass_corretta[] = "password123";
11
12    return (strcmp(username, user_corretto) == 0 &&
13            strcmp(password, pass_corretta) == 0);
14 }
15
16 int main() {
17     char username[MAX_USERNAME];
18     char password[MAX_PASSWORD];
19
20     printf("=== SISTEMA DI LOGIN ===\n");
21     printf("Username: ");
22     scanf("%s", username);
23
24     printf("Password: ");
25     scanf("%s", password);
26
27     if (verifica_login(username, password)) {
28         printf("\nAccesso consentito!\n");
29     } else {
30         printf("\nCredenziali non valide!\n");
31     }
32
33     return 0;
34 }
```

### 8.10.2 Analisi di Testo

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 void analizza_testo(char *testo) {
6     int lettere = 0, cifre = 0, spazi = 0, altri = 0;
7
8     for (int i = 0; testo[i] != '\0'; i++) {
9         if (isalpha(testo[i])) {
10             lettere++;
11         } else if (isdigit(testo[i])) {
12             cifre++;
13         } else if (isspace(testo[i])) {
14             spazi++;
15         } else {
16             altri++;
17         }
18     }
19
20     printf("=== ANALISI TESTO ===\n");
21     printf("Lunghezza: %lu\n", strlen(testo));
22     printf("Lettere: %d\n", lettere);
23     printf("Cifre: %d\n", cifre);
24     printf("Spazi: %d\n", spazi);
25     printf("Altri caratteri: %d\n", altri);
26 }
```

```
26 }
27
28 int main() {
29     char testo[200];
30
31     printf("Inserisci un testo:\n");
32     fgets(testo, sizeof(testo), stdin);
33
34     // Rimuovi newline finale se presente
35     testo[strcspn(testo, "\n")] = '\0';
36
37     analizza_testo(testo);
38
39     return 0;
40 }
```

## 8.11 Esercizi

### 8.11.1 Livello Base

1. Scrivi una funzione che conta quante vocali ci sono in una stringa.
2. Crea una funzione che verifica se una stringa contiene solo caratteri alfabetici.
3. Implementa una funzione che rimuove tutti i caratteri non alfanumerici da una stringa.
4. Scrivi un programma che legge una frase e stampa ogni parola su una riga separata.

### 8.11.2 Livello Intermedio

1. Implementa una funzione che sostituisce tutte le occorrenze di una sottostringa con un'altra.
2. Crea una funzione che verifica se due stringhe sono anagrammi.
3. Scrivi un programma che ordina un array di stringhe in ordine alfabetico.
4. Implementa una funzione che comprime una stringa (es: "aaabbc" → "a3b2c1").

### 8.11.3 Livello Avanzato

1. Implementa un parser di espressioni matematiche semplici (es: "3 + 5 \* 2").
2. Crea un programma che valida indirizzi email usando analisi di stringhe.
3. Scrivi una funzione che implementa il pattern matching semplice (con wildcard \* e ?).
4. Implementa un sistema di cifratura/decifratura Caesar per stringhe.

# Capitolo 9

## Struct

### 9.1 Obiettivi di Apprendimento

Al termine di questo capitolo sarai in grado di:

- Definire struct per raggruppare dati correlati
- Dichiarare e inizializzare variabili di tipo struct
- Accedere ai membri di una struct usando l'operatore punto (.)
- Utilizzare puntatori a struct e l'operatore freccia (->) per accedere ai membri
- Creare alias per struct con la parola chiave typedef
- Gestire array di struct
- Implementare struct annidate per rappresentare dati complessi
- Passare struct a funzioni per valore e per riferimento (tramite puntatori)
- Allocare struct dinamicamente con la funzione malloc() della libreria stdlib

### 9.2 Introduzione alle Struct

Le struct (strutture) permettono di raggruppare variabili di tipi diversi sotto un unico nome, creando nuovi tipi di dati personalizzati.

#### 9.2.1 Perché Usare le Struct?

- Organizzare dati correlati insieme
- Creare tipi di dati complessi
- Migliorare la leggibilità del codice
- Facilitare la gestione di dati strutturati

**Nota**

Le struct sono fondamentali per la programmazione strutturata e permettono di modellare entità del mondo reale nel codice.

## 9.3 Definizione di una Struct

### 9.3.1 Sintassi Base

```
1 struct nome_struct {  
2     tipo1 campo1;  
3     tipo2 campo2;  
4     // ... altri campi  
5 };
```

### 9.3.2 Esempi di Definizione

```
1 #include <stdio.h>  
2  
3 // Struct per rappresentare un punto 2D  
4 struct Punto {  
5     int x;  
6     int y;  
7 };  
8  
9 // Struct per rappresentare una data  
10 struct Data {  
11     int giorno;  
12     int mese;  
13     int anno;  
14 };  
15  
16 // Struct per rappresentare uno studente  
17 struct Studente {  
18     char nome[50];  
19     char cognome[50];  
20     int matricola;  
21     float media;  
22 };  
23  
24 int main(int argc, char** argv) {  
25     // Dichiarazione di variabili struct  
26     struct Punto p1;  
27     struct Data oggi;  
28     struct Studente s1;  
29  
30     return 0;  
31 }
```

## 9.4 Dichiarazione e Inizializzazione

### 9.4.1 Dichiarazione

```
1 #include <stdio.h>
2
3 struct Persona {
4     char nome[30];
5     int eta;
6     float altezza;
7 };
8
9 int main(int argc, char** argv) {
10     // Dichiarazione semplice
11     struct Persona p1;
12
13     // Dichiarazione multipla
14     struct Persona p2, p3, p4;
15
16     return 0;
17 }
```

### 9.4.2 Inizializzazione

```
1 #include <stdio.h>
2
3 struct Punto {
4     int x;
5     int y;
6 };
7
8 int main(int argc, char** argv) {
9     // Inizializzazione alla dichiarazione
10    struct Punto p1 = {10, 20};
11
12    // Inizializzazione con nomi dei campi (C99)
13    struct Punto p2 = {.x = 5, .y = 15};
14
15    // Inizializzazione parziale
16    struct Punto p3 = {10}; // x=10, y=0
17
18    // Inizializzazione a zero
19    struct Punto p4 = {0}; // x=0, y=0
20
21    printf("p1: (%d, %d)\n", p1.x, p1.y); // Accesso ai campi p1.x e
22    printf("p2: (%d, %d)\n", p2.x, p2.y); // Accesso ai campi p2.x e
23    printf("p3: (%d, %d)\n", p3.x, p3.y); // Accesso ai campi p3.x e
24    printf("p4: (%d, %d)\n", p4.x, p4.y); // Accesso ai campi p4.x e
25
26    return 0;
27 }
```

## 9.5 Accesso ai Membri

### 9.5.1 Operatore Punto (.)

```
1 #include <stdio.h>
2 #include <string.h>
3
4 struct Studente {
5     char nome[30];
6     int eta;
7     float media;
8 };
9
10 int main(int argc, char** argv) {
11     struct Studente s1;
12
13     // Assegnamento ai membri della struct Studente
14     strcpy(s1.nome, "Mario Rossi"); // Assegna al campo s1.nome della
        struct Studente
15     s1.eta = 20;                      // Assegna al campo s1.eta della
        struct Studente
16     s1.media = 27.5;                  // Assegna al campo s1.media
        della struct Studente
17
18     // Lettura dei membri della struct Studente
19     printf("Nome: %s\n", s1.nome);    // Accesso al campo s1.nome della
        struct Studente
20     printf("Eta': %d\n", s1.eta);      // Accesso al campo s1.eta della
        struct Studente
21     printf("Media: %.1f\n", s1.media); // Accesso al campo s1.media
        della struct Studente
22
23     return 0;
24 }
```

### 9.5.2 Input da Utente

```
1 #include <stdio.h>
2
3 struct Libro {
4     char titolo[100];
5     char autore[50];
6     int anno;
7     float prezzo;
8 };
9
10 int main(int argc, char** argv) {
11     struct Libro libro;
12
13     printf("=== INSERIMENTO LIBRO ===\n");
14
15     printf("Titolo: ");
16     fgets(libro.titolo, sizeof(libro.titolo), stdin); // Legge nel
        campo libro.titolo della struct Libro
17
18     printf("Autore: ");
```

```

19     fgets(libro.autore, sizeof(libro.autore), stdin); // Legge nel
        campo libro.autore della struct Libro
20
21     printf("Anno: ");
22     scanf("%d", &libro.anno); // Legge nel campo libro.anno della
        struct Libro
23
24     printf("Prezzo: ");
25     scanf("%f", &libro.prezzo); // Legge nel campo libro.prezzo della
        struct Libro
26
27     printf("\n=== DATI LIBRO ===\n");
28     printf("Titolo: %s", libro.titolo); // Stampa il campo libro.
        titolo della struct Libro
29     printf("Autore: %s", libro.autore); // Stampa il campo libro.
        autore della struct Libro
30     printf("Anno: %d\n", libro.anno); // Stampa il campo libro.
        anno della struct Libro
31     printf("Prezzo: %.2f euro\n", libro.prezzo); // Stampa il campo
        libro.prezzo della struct Libro
32
33     return 0;
34 }

```

## 9.6 Typedef

typedef permette di creare alias per i tipi, semplificando la sintassi.

```

1  #include <stdio.h>
2
3  // Senza typedef
4  struct Punto {
5      int x;
6      int y;
7  };
8
9  // Con typedef (metodo 1)
10 typedef struct {
11     int x;
12     int y;
13 } Punto2D;
14
15 // Con typedef (metodo 2)
16 typedef struct Rettangolo {
17     int larghezza;
18     int altezza;
19 } Rettangolo;
20
21 int main(int argc, char** argv) {
22     // Senza typedef
23     struct Punto p1 = {10, 20};
24
25     // Con typedef (piu' semplice)
26     Punto2D p2 = {30, 40};
27     Rettangolo r1 = {100, 50};
28 }

```

```

29     printf("p1: (%d, %d)\n", p1.x, p1.y); // Accesso ai campi p1.x e
        p1.y della struct Punto
30     printf("p2: (%d, %d)\n", p2.x, p2.y); // Accesso ai campi p2.x e
        p2.y della struct Punto2D
31     printf("r1: %dx%d\n", r1.larghezza, r1.altezza); // Accesso ai
        campi r1.larghezza e r1.altezza della struct Rettangolo
32
33     return 0;
34 }

```

## 9.7 Array di Struct

```

1  #include <stdio.h>
2  #include <string.h>
3
4  typedef struct {
5      char nome[30];
6      int eta;
7      float voto;
8  } Studente;
9
10 int main(int argc, char** argv) {
11     // Array di 3 studenti
12     Studente classe[3];
13
14     // Inizializzazione degli elementi dell'array classe
15     strcpy(classe[0].nome, "Mario"); // Assegna al campo classe[0].
        nome della struct Studente
16     classe[0].eta = 20; // Assegna al campo classe[0].
        eta della struct Studente
17     classe[0].voto = 28.5; // Assegna al campo classe[0].
        voto della struct Studente
18
19     strcpy(classe[1].nome, "Lucia"); // Assegna al campo classe[1].
        nome della struct Studente
20     classe[1].eta = 19; // Assegna al campo classe[1].
        eta della struct Studente
21     classe[1].voto = 30.0; // Assegna al campo classe[1].
        voto della struct Studente
22
23     strcpy(classe[2].nome, "Giovanni"); // Assegna al campo classe
        [2].nome della struct Studente
24     classe[2].eta = 21; // Assegna al campo classe
        [2].eta della struct Studente
25     classe[2].voto = 26.0; // Assegna al campo classe
        [2].voto della struct Studente
26
27     // Stampa di tutti gli studenti dell'array classe
28     printf("=== ELENCO STUDENTI ===\n");
29     for (int i = 0; i < 3; i++) {
30         printf("%d. %s - Eta': %d - Voto: %.1f\n",
31             i + 1, classe[i].nome, classe[i].eta, classe[i].voto);
32         // Accesso ai campi della struct Studente
33     }
34
35     // Calcolo della media dei voti degli studenti

```

```

35     float media_voti = 0;
36     for (int i = 0; i < 3; i++) {
37         media_voti += classe[i].voto;    // Somma i campi classe[i].voto
                                           delle struct Studente
38     }
39     media_voti /= 3;
40     printf("\nMedia dei voti: %.2f\n", media_voti);
41
42     return 0;
43 }

```

## 9.8 Struct Annidate

Le struct possono contenere altre struct come membri.

```

1  #include <stdio.h>
2  #include <string.h>
3
4  typedef struct {
5      int giorno;
6      int mese;
7      int anno;
8  } Data;
9
10 typedef struct {
11     char via[50];
12     char citta[30];
13     int cap;
14 } Indirizzo;
15
16 typedef struct {
17     char nome[30];
18     char cognome[30];
19     Data data_nascita;
20     Indirizzo indirizzo;
21 } Persona;
22
23 int main(int argc, char** argv) {
24     Persona p1;
25
26     // Assegnamento ai campi della struct Persona e delle struct
27     // annidate
28     strcpy(p1.nome, "Mario");           // Assegna al campo p1.nome della
                                           struct Persona
29     strcpy(p1.cognome, "Rossi");        // Assegna al campo p1.cognome
                                           della struct Persona
30
31     p1.data_nascita.giorno = 15;        // Assegna al campo p1.data_nascita
                                           .giorno della struct Data annidata
32     p1.data_nascita.mese = 3;           // Assegna al campo p1.data_nascita
                                           .mese della struct Data annidata
33     p1.data_nascita.anno = 2000;        // Assegna al campo p1.data_nascita
                                           .anno della struct Data annidata
34
35     strcpy(p1.indirizzo.via, "Via Roma 123"); // Assegna al campo p1.
                                           indirizzo.via della struct Indirizzo annidata

```

```

35 strcpy(p1.indirizzo.citta, "Milano");           // Assegna al campo p1.
    indirizzo.citta della struct Indirizzo annidata
36 p1.indirizzo.cap = 20100;                       // Assegna al campo p1.
    indirizzo.cap della struct Indirizzo annidata
37
38 // Stampa dei dati della struct Persona
39 printf("=== DATI PERSONA ===\n");
40 printf("Nome: %s %s\n", p1.nome, p1.cognome);    // Accesso ai campi
    p1.nome e p1.cognome della struct Persona
41 printf("Data di nascita: %02d/%02d/%04d\n",
42        p1.data_nascita.giorno,                  // Accesso al campo p1.
    data_nascita.giorno della struct Data annidata
43        p1.data_nascita.mese,                    // Accesso al campo p1.
    data_nascita.mese della struct Data annidata
44        p1.data_nascita.anno);                   // Accesso al campo p1.
    data_nascita.anno della struct Data annidata
45 printf("Indirizzo: %s, %s (%d)\n",
46        p1.indirizzo.via,                        // Accesso al campo p1.indirizzo.
    via della struct Indirizzo annidata
47        p1.indirizzo.citta,                      // Accesso al campo p1.indirizzo.
    citta della struct Indirizzo annidata
48        p1.indirizzo.cap);                       // Accesso al campo p1.indirizzo.
    cap della struct Indirizzo annidata
49
50 return 0;
51 }

```

## 9.9 Puntatori a Struct

### 9.9.1 Operatore Freccia (->)

```

1  #include <stdio.h>
2  #include <string.h>
3
4  typedef struct {
5      char titolo[50];
6      int anno;
7      float rating;
8  } Film;
9
10 int main(int argc, char** argv) {
11     Film f1;
12     Film *ptr = &f1;
13
14     // Accesso con operatore punto (tramite variabile struct)
15     strcpy(f1.titolo, "Il Padrino"); // Assegna al campo f1.titolo
    della struct Film
16     f1.anno = 1972;                  // Assegna al campo f1.anno
    della struct Film
17     f1.rating = 9.2;                  // Assegna al campo f1.rating
    della struct Film
18
19     // Accesso con operatore freccia (tramite puntatore a struct)
20     printf("Titolo: %s\n", ptr->titolo); // Accesso al campo ptr->
    titolo tramite puntatore alla struct Film

```

```

21     printf("Anno: %d\n", ptr->anno);           // Accesso al campo ptr->
        anno tramite puntatore alla struct Film
22     printf("Rating: %.1f\n", ptr->rating); // Accesso al campo ptr->
        rating tramite puntatore alla struct Film
23
24     // Modifica tramite puntatore a struct
25     ptr->anno = 1973;    // Modifica il campo ptr->anno tramite
        puntatore alla struct Film
26     ptr->rating = 9.5;  // Modifica il campo ptr->rating tramite
        puntatore alla struct Film
27
28     printf("\nDopo modifica:\n");
29     printf("Anno: %d\n", f1.anno);           // Accesso al campo f1.anno
        della struct Film
30     printf("Rating: %.1f\n", f1.rating); // Accesso al campo f1.rating
        della struct Film
31
32     return 0;
33 }

```

### Nota

`ptr->campo` è equivalente a `(*ptr).campo`, ma la notazione con la freccia è più leggibile.

## 9.10 Struct e Funzioni

### 9.10.1 Passaggio per Valore

```

1  #include <stdio.h>
2
3  typedef struct {
4      int x;
5      int y;
6  } Punto;
7
8  // Passaggio per valore (copia della struct)
9  void stampa_punto(Punto p) {
10     printf("Punto: (%d, %d)\n", p.x, p.y); // Accesso ai campi p.x e
        p.y della struct Punto
11 }
12
13 // Modifica non influenza l'originale (passaggio per valore)
14 void sposta(Punto p, int dx, int dy) {
15     p.x += dx; // Modifica locale del campo p.x della struct Punto (
        copia)
16     p.y += dy; // Modifica locale del campo p.y della struct Punto (
        copia)
17     printf("Dentro sposta: (%d, %d)\n", p.x, p.y); // Accesso ai
        campi p.x e p.y della struct Punto
18 }
19
20 int main(int argc, char** argv) {
21     Punto p1 = {10, 20};
22
23     stampa_punto(p1);

```

```
24     sposta(p1, 5, 10);
25     stampa_punto(p1); // Punto ancora (10, 20)
26
27     return 0;
28 }
```

### 9.10.2 Passaggio per Riferimento

```
1  #include <stdio.h>
2
3  typedef struct {
4      int x;
5      int y;
6  } Punto;
7
8  // Passaggio per riferimento (tramite puntatore a struct)
9  void sposta(Punto *p, int dx, int dy) {
10     p->x += dx; // Modifica il campo p->x tramite puntatore alla
11                // struct Punto
12     p->y += dy; // Modifica il campo p->y tramite puntatore alla
13                // struct Punto
14 }
15
16 void stampa_punto(Punto *p) {
17     printf("Punto: (%d, %d)\n", p->x, p->y); // Accesso ai campi p->x
18     // e p->y tramite puntatore alla struct Punto
19 }
20
21 int main(int argc, char** argv) {
22     Punto p1 = {10, 20};
23
24     printf("Originale: ");
25     stampa_punto(&p1);
26
27     sposta(&p1, 5, 10);
28
29     printf("Dopo spostamento: ");
30     stampa_punto(&p1); // Punto modificato: (15, 30)
31
32     return 0;
33 }
```

### 9.10.3 Ritorno di Struct

```
1  #include <stdio.h>
2  #include <math.h>
3
4  typedef struct {
5      float x;
6      float y;
7  } Punto;
8
9  // Funzione che restituisce una struct Punto
10 Punto crea_punto(float x, float y) {
11     Punto p;
```

```

12     p.x = x; // Assegna al campo p.x della struct Punto
13     p.y = y; // Assegna al campo p.y della struct Punto
14     return p;
15 }
16
17 float distanza(Punto p1, Punto p2) {
18     float dx = p2.x - p1.x; // Calcola la differenza usando i campi
19                             // p2.x e p1.x delle struct Punto
20     float dy = p2.y - p1.y; // Calcola la differenza usando i campi
21                             // p2.y e p1.y delle struct Punto
22     return sqrt(dx * dx + dy * dy); // La funzione sqrt() è della
23                                     // libreria math
24 }
25
26 Punto punto_medio(Punto p1, Punto p2) {
27     Punto medio;
28     medio.x = (p1.x + p2.x) / 2; // Calcola la media dei campi p1.x e
29                                 // p2.x e la assegna a medio.x della struct Punto
30     medio.y = (p1.y + p2.y) / 2; // Calcola la media dei campi p1.y e
31                                 // p2.y e la assegna a medio.y della struct Punto
32     return medio;
33 }
34
35 int main(int argc, char** argv) {
36     Punto p1 = crea_punto(0, 0);
37     Punto p2 = crea_punto(3, 4);
38
39     printf("p1: (%.1f, %.1f)\n", p1.x, p1.y); // Accesso ai campi p1.
40                                                // x e p1.y della struct Punto
41     printf("p2: (%.1f, %.1f)\n", p2.x, p2.y); // Accesso ai campi p2.
42                                                // x e p2.y della struct Punto
43
44     float dist = distanza(p1, p2);
45     printf("Distanza: %.2f\n", dist);
46
47     Punto medio = punto_medio(p1, p2);
48     printf("Punto medio: (%.1f, %.1f)\n", medio.x, medio.y); //
49                                                                // Accesso ai campi medio.x e medio.y della struct Punto
50
51     return 0;
52 }

```

## 9.11 Allocazione Dinamica di Struct

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 typedef struct {
6     char nome[30];
7     int eta;
8 } Persona;
9
10 int main(int argc, char** argv) {
11     // Allocazione dinamica di una singola struct con malloc() della
12     // libreria stdlib

```

```
12  Persona *p1 = (Persona*)malloc(sizeof(Persona));
13
14  if (p1 == NULL) {
15      printf("Errore di allocazione\n");
16      return 1;
17  }
18
19  strcpy(p1->nome, "Mario"); // Assegna al campo p1->nome tramite
                             // puntatore alla struct Persona
20  p1->eta = 25;               // Assegna al campo p1->eta tramite
                             // puntatore alla struct Persona
21
22  printf("Nome: %s, Eta': %d\n", p1->nome, p1->eta); // Accesso ai
                             // campi tramite puntatore alla struct Persona
23
24  free(p1); // La funzione free() della libreria stdlib libera la
             // memoria allocata
25
26  // Allocazione dinamica di un array di struct con malloc() della
             // libreria stdlib
27  int n = 3;
28  Persona *persone = (Persona*)malloc(n * sizeof(Persona));
29
30  if (persone == NULL) {
31      printf("Errore di allocazione\n");
32      return 1;
33  }
34
35  // Inizializzazione degli elementi dell'array di struct
36  strcpy(persone[0].nome, "Lucia"); // Assegna al campo persone[0].
                                     // nome della struct Persona
37  persone[0].eta = 22;              // Assegna al campo persone[0].
                                     // eta della struct Persona
38
39  strcpy(persone[1].nome, "Giovanni"); // Assegna al campo persone
                                     // [1].nome della struct Persona
40  persone[1].eta = 28;              // Assegna al campo persone
                                     // [1].eta della struct Persona
41
42  strcpy(persone[2].nome, "Anna"); // Assegna al campo persone[2].
                                     // nome della struct Persona
43  persone[2].eta = 30;              // Assegna al campo persone[2].
                                     // eta della struct Persona
44
45  // Stampa dell'array di struct
46  printf("\nElenco persone:\n");
47  for (int i = 0; i < n; i++) {
48      printf("%d. %s - %d anni\n",
49              i + 1, persone[i].nome, persone[i].eta); // Accesso ai
                                                         // campi delle struct Persona
50  }
51
52  free(persone); // La funzione free() della libreria stdlib libera
                 // la memoria allocata
53
54  return 0;
55 }
```

## 9.12 Esempi Pratici

### 9.12.1 Sistema di Gestione Studenti

```

1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAX_STUDENTI 50
5
6  typedef struct {
7      int matricola;
8      char nome[30];
9      char cognome[30];
10     float media;
11 } Studente;
12
13 typedef struct {
14     Studente studenti[MAX_STUDENTI];
15     int numero_studenti;
16 } Database;
17
18 void aggiungi_studente(Database *db, Studente s) {
19     if (db->numero_studenti < MAX_STUDENTI) { // Accesso al campo db
20         // db->numero_studenti tramite puntatore alla struct Database
21         db->studenti[db->numero_studenti++] = s; // Accesso all'array
22         // db->studenti tramite puntatore alla struct Database
23         printf("Studente aggiunto con successo!\n");
24     } else {
25         printf("Database pieno!\n");
26     }
27 }
28
29 void stampa_studenti(Database *db) {
30     printf("\n=== ELENCO STUDENTI ===\n");
31     for (int i = 0; i < db->numero_studenti; i++) { // Accesso al
32         // campo db->numero_studenti tramite puntatore alla struct
33         // Database
34         Studente s = db->studenti[i]; // Accesso all'array db->
35         // studenti tramite puntatore alla struct Database
36         printf("%d. Matricola: %d - %s %s - Media: %.2f\n",
37             i + 1, s.matricola, s.nome, s.cognome, s.media); //
38         // Accesso ai campi della struct Studente
39     }
40 }
41
42 Studente* cerca_per_matricola(Database *db, int matricola) {
43     for (int i = 0; i < db->numero_studenti; i++) { // Accesso al
44         // campo db->numero_studenti tramite puntatore alla struct
45         // Database
46         if (db->studenti[i].matricola == matricola) { // Accesso al
47             // campo db->studenti[i].matricola della struct Studente
48             return &db->studenti[i]; // Ritorna l'indirizzo dell'
49             // elemento dell'array db->studenti
50         }
51     }
52     return NULL;
53 }
54

```

```

45 int main(int argc, char** argv) {
46     Database db = {.numero_studenti = 0};
47
48     // Aggiungi studenti
49     Studente s1 = {12345, "Mario", "Rossi", 27.5};
50     Studente s2 = {12346, "Lucia", "Bianchi", 29.0};
51     Studente s3 = {12347, "Giovanni", "Verdi", 26.5};
52
53     aggiungi_studente(&db, s1);
54     aggiungi_studente(&db, s2);
55     aggiungi_studente(&db, s3);
56
57     stampa_studenti(&db);
58
59     // Cerca studente
60     int matricola_cerca = 12346;
61     Studente *trovato = cerca_per_matricola(&db, matricola_cerca);
62
63     if (trovato != NULL) {
64         printf("\nStudente trovato:\n");
65         printf("Matricola: %d - %s %s - Media: %.2f\n",
66             trovato->matricola, trovato->nome, // Accesso ai campi
67             // tramite puntatore alla struct Studente
68             trovato->cognome, trovato->media); // Accesso ai campi
69             // tramite puntatore alla struct Studente
70     } else {
71         printf("\nStudente non trovato.\n");
72     }
73
74     return 0;
75 }

```

### 9.12.2 Sistema di Coordinate 3D

```

1  #include <stdio.h>
2  #include <math.h>
3
4  typedef struct {
5      float x;
6      float y;
7      float z;
8  } Punto3D;
9
10 Punto3D crea_punto(float x, float y, float z) {
11     Punto3D p = {x, y, z};
12     return p;
13 }
14
15 void stampa_punto(Punto3D p) {
16     printf("(%.2f, %.2f, %.2f)", p.x, p.y, p.z); // Accesso ai campi
17     // p.x, p.y e p.z della struct Punto3D
18 }
19
20 float distanza(Punto3D p1, Punto3D p2) {
21     float dx = p2.x - p1.x; // Calcola la differenza usando i campi
22     // p2.x e p1.x delle struct Punto3D

```

```

21     float dy = p2.y - p1.y; // Calcola la differenza usando i campi
    p2.y e p1.y delle struct Punto3D
22     float dz = p2.z - p1.z; // Calcola la differenza usando i campi
    p2.z e p1.z delle struct Punto3D
23     return sqrt(dx*dx + dy*dy + dz*dz); // La funzione sqrt() è della
    libreria math
24 }
25
26 Punto3D somma_vettori(Punto3D v1, Punto3D v2) {
27     Punto3D risultato;
28     risultato.x = v1.x + v2.x; // Somma i campi v1.x e v2.x e assegna
    a risultato.x della struct Punto3D
29     risultato.y = v1.y + v2.y; // Somma i campi v1.y e v2.y e assegna
    a risultato.y della struct Punto3D
30     risultato.z = v1.z + v2.z; // Somma i campi v1.z e v2.z e assegna
    a risultato.z della struct Punto3D
31     return risultato;
32 }
33
34 float prodotto_scalare(Punto3D v1, Punto3D v2) {
35     return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z; // Calcola il
    prodotto scalare usando i campi delle struct Punto3D
36 }
37
38 int main(int argc, char** argv) {
39     Punto3D p1 = crea_punto(1, 2, 3);
40     Punto3D p2 = crea_punto(4, 5, 6);
41
42     printf("p1 = ");
43     stampa_punto(p1);
44     printf("\n");
45
46     printf("p2 = ");
47     stampa_punto(p2);
48     printf("\n");
49
50     printf("\nDistanza: %.2f\n", distanza(p1, p2));
51
52     Punto3D somma = somma_vettori(p1, p2);
53     printf("Somma: ");
54     stampa_punto(somma);
55     printf("\n");
56
57     printf("Prodotto scalare: %.2f\n", prodotto_scalare(p1, p2));
58
59     return 0;
60 }

```

## 9.13 Esercizi

### 9.13.1 Livello Base

1. Definisci una struct **Rettangolo** con base e altezza, e scrivi funzioni per calcolare area e perimetro.
2. Crea una struct **Tempo** (ore, minuti, secondi) e una funzione che converte in secondi totali.

3. Implementa una struct **ContoCorrente** con saldo e operazioni di deposito/prelievo.
4. Scrivi un programma con struct **Prodotto** (nome, prezzo, quantità) per gestire un inventario.

### 9.13.2 Livello Intermedio

1. Crea un sistema di rubrica telefonica con array di struct **Contatto**.
2. Implementa una struct **Matrice2D** con allocazione dinamica e operazioni base.
3. Scrivi un programma che gestisce un database di libri con ricerca e ordinamento.
4. Crea struct per rappresentare poligoni e calcolare perimetro e area.

### 9.13.3 Livello Avanzato

1. Implementa una lista collegata usando struct con puntatori.
2. Crea un sistema di gestione di un parcheggio con struct per veicoli e posti.
3. Implementa un grafo usando struct per nodi e archi.
4. Scrivi un programma che simula un sistema bancario completo con conti, transazioni e storici.

# Capitolo 10

## Gestione dei File

### 10.1 Introduzione ai File

La gestione dei file permette di salvare dati in modo permanente su disco e di leggerli in seguito. In C, i file vengono gestiti attraverso puntatori a `FILE`.

#### 10.1.1 Tipi di File

- **File di testo:** contengono caratteri leggibili (es: .txt, .csv)
- **File binari:** contengono dati in formato binario (es: immagini, eseguibili)

#### Nota

Per gestire i file in C è necessario includere la libreria `stdio.h`.

### 10.2 Apertura e Chiusura di File

#### 10.2.1 Funzione fopen

```
1 FILE *fopen(const char *nome_file, const char *modalita);
```

#### 10.2.2 Modalità di Apertura

- `"r"`: lettura (read) - il file deve esistere
- `"w"`: scrittura (write) - crea nuovo file o sovrascrive
- `"a"`: append - aggiunge in coda al file
- `"r+"`: lettura e scrittura - il file deve esistere
- `"w+"`: lettura e scrittura - crea nuovo file
- `"a+"`: lettura e append
- `"rb"`, `"wb"`, ecc.: modalità binaria

### 10.2.3 Esempio Base

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     FILE *file;
6
7     // Apertura file
8     file = fopen("test.txt", "w");
9
10    // Controlla se l'apertura e' riuscita
11    if (file == NULL) {
12        printf("Errore nell'apertura del file!\n");
13        return 1;
14    }
15
16    printf("File aperto con successo!\n");
17
18    // Chiusura file
19    fclose(file);
20
21    return 0;
22 }
```

#### Attenzione

Controlla sempre che `fopen()` non restituisca `NULL` prima di usare il file. Ricorda sempre di chiudere i file con `fclose()`.

## 10.3 Scrittura su File

### 10.3.1 fprintf

Scrivi dati formattati su file (simile a `printf`).

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("output.txt", "w");
5
6     if (file == NULL) {
7         printf("Errore apertura file!\n");
8         return 1;
9     }
10
11    // Scrittura formattata
12    fprintf(file, "Questa e' la prima riga\n");
13    fprintf(file, "Numero: %d\n", 42);
14    fprintf(file, "Float: %.2f\n", 3.14159);
15
16    fclose(file);
17    printf("Dati scritti su output.txt\n");
18
19    return 0;
20 }
```

```
20 }
```

### 10.3.2 fputs

Scrivi una stringa su file.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("testo.txt", "w");
5
6     if (file == NULL) {
7         printf("Errore!\n");
8         return 1;
9     }
10
11     fputs("Prima riga\n", file);
12     fputs("Seconda riga\n", file);
13     fputs("Terza riga\n", file);
14
15     fclose(file);
16     return 0;
17 }
```

### 10.3.3 fputc

Scrivi un singolo carattere.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("caratteri.txt", "w");
5
6     if (file == NULL) {
7         printf("Errore!\n");
8         return 1;
9     }
10
11     // Scrive caratteri singoli
12     fputc('H', file);
13     fputc('e', file);
14     fputc('l', file);
15     fputc('l', file);
16     fputc('o', file);
17     fputc('\n', file);
18
19     fclose(file);
20     return 0;
21 }
```

## 10.4 Lettura da File

### 10.4.1 fscanf

Legge dati formattati da file.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("dati.txt", "r");
5
6     if (file == NULL) {
7         printf("File non trovato!\n");
8         return 1;
9     }
10
11     int numero;
12     float decimale;
13     char parola[50];
14
15     // Legge dati formattati
16     fscanf(file, "%d", &numero);
17     fscanf(file, "%f", &decimale);
18     fscanf(file, "%s", parola);
19
20     printf("Numero: %d\n", numero);
21     printf("Decimale: %.2f\n", decimale);
22     printf("Parola: %s\n", parola);
23
24     fclose(file);
25     return 0;
26 }
```

### 10.4.2 fgets

Legge una riga da file.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("testo.txt", "r");
5
6     if (file == NULL) {
7         printf("File non trovato!\n");
8         return 1;
9     }
10
11     char linea[200];
12
13     // Legge riga per riga
14     while (fgets(linea, sizeof(linea), file) != NULL) {
15         printf("%s", linea);
16     }
17
18     fclose(file);
19     return 0;
20 }
```

### 10.4.3 fgetc

Legge un carattere alla volta.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("testo.txt", "r");
5
6     if (file == NULL) {
7         printf("File non trovato!\n");
8         return 1;
9     }
10
11     char c;
12
13     // Legge carattere per carattere
14     while ((c = fgetc(file)) != EOF) {
15         putchar(c);
16     }
17
18     fclose(file);
19     return 0;
20 }
```

## 10.5 Controllo Fine File e Errori

### 10.5.1 feof e ferror

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("dati.txt", "r");
5
6     if (file == NULL) {
7         perror("Errore apertura file");
8         return 1;
9     }
10
11     int num;
12
13     while (1) {
14         int result = fscanf(file, "%d", &num);
15
16         // Controlla fine file
17         if (feof(file)) {
18             printf("Fine del file raggiunta\n");
19             break;
20         }
21
22         // Controlla errori
23         if (ferror(file)) {
```

```
24         printf("Errore di lettura!\n");
25         break;
26     }
27
28     // Controlla risultato fscanf
29     if (result != 1) {
30         printf("Formato non valido\n");
31         break;
32     }
33
34     printf("Numero letto: %d\n", num);
35 }
36
37 fclose(file);
38 return 0;
39 }
```

## 10.6 Posizionamento nel File

### 10.6.1 fseek

Sposta il puntatore del file in una posizione specifica.

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      FILE *file = fopen("test.txt", "r");
5
6      if (file == NULL) {
7          printf("Errore!\n");
8          return 1;
9      }
10
11     // Vai all'inizio del file
12     fseek(file, 0, SEEK_SET);
13
14     // Vai alla fine del file
15     fseek(file, 0, SEEK_END);
16
17     // Ottieni la dimensione del file
18     long dimensione = ftell(file);
19     printf("Dimensione del file: %ld byte\n", dimensione);
20
21     // Torna all'inizio
22     rewind(file); // Equivalente a fseek(file, 0, SEEK_SET)
23
24     fclose(file);
25     return 0;
26 }
```

### 10.6.2 ftell

Restituisce la posizione corrente nel file.

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     FILE *file = fopen("test.txt", "r");
5
6     if (file == NULL) {
7         printf("Errore!\n");
8         return 1;
9     }
10
11     printf("Posizione iniziale: %ld\n", ftell(file));
12
13     fgetc(file); // Legge un carattere
14     printf("Dopo 1 carattere: %ld\n", ftell(file));
15
16     fgetc(file);
17     printf("Dopo 2 caratteri: %ld\n", ftell(file));
18
19     fclose(file);
20     return 0;
21 }
```

## 10.7 File Binari

### 10.7.1 fwrite

Scrivi dati binari su file.

```
1 #include <stdio.h>
2
3 typedef struct {
4     int id;
5     char nome[30];
6     float voto;
7 } Studente;
8
9 int main(int argc, char** argv) {
10     FILE *file = fopen("studenti.dat", "wb");
11
12     if (file == NULL) {
13         printf("Errore!\n");
14         return 1;
15     }
16
17     Studente s1 = {1, "Mario Rossi", 27.5};
18     Studente s2 = {2, "Lucia Bianchi", 29.0};
19
20     // Scrivi struct binarie
21     fwrite(&s1, sizeof(Studente), 1, file);
22     fwrite(&s2, sizeof(Studente), 1, file);
23
24     fclose(file);
25     printf("Dati scritti in formato binario\n");
26
27     return 0;
28 }
```

```
28 }
```

## 10.7.2 fread

Legge dati binari da file.

```
1  #include <stdio.h>
2
3  typedef struct {
4      int id;
5      char nome[30];
6      float voto;
7  } Studente;
8
9  int main(int argc, char** argv) {
10     FILE *file = fopen("studenti.dat", "rb");
11
12     if (file == NULL) {
13         printf("File non trovato!\n");
14         return 1;
15     }
16
17     Studente s;
18
19     // Leggi struct dal file
20     while (fread(&s, sizeof(Studente), 1, file) == 1) {
21         printf("ID: %d\n", s.id);
22         printf("Nome: %s\n", s.nome);
23         printf("Voto: %.1f\n\n", s.voto);
24     }
25
26     fclose(file);
27     return 0;
28 }
```

## 10.8 Esempi Pratici

### 10.8.1 Copia di File

```
1  #include <stdio.h>
2
3  int main(int argc, char** argv) {
4      FILE *sorgente, *destinazione;
5      char ch;
6
7      sorgente = fopen("originale.txt", "r");
8      if (sorgente == NULL) {
9          printf("Impossibile aprire il file sorgente!\n");
10         return 1;
11     }
12
13     destinazione = fopen("copia.txt", "w");
14     if (destinazione == NULL) {
15         printf("Impossibile creare il file destinazione!\n");
```

```
16     fclose(sorgente);
17     return 1;
18 }
19
20 // Copia carattere per carattere
21 while ((ch = fgetc(sorgente)) != EOF) {
22     fputc(ch, destinazione);
23 }
24
25 printf("File copiato con successo!\n");
26
27 fclose(sorgente);
28 fclose(destinazione);
29
30 return 0;
31 }
```

### 10.8.2 Conta Righe, Parole e Caratteri

```
1 #include <stdio.h>
2 #include <ctype.h>
3
4 int main(int argc, char** argv) {
5     FILE *file = fopen("testo.txt", "r");
6
7     if (file == NULL) {
8         printf("File non trovato!\n");
9         return 1;
10    }
11
12    int righe = 0, parole = 0, caratteri = 0;
13    int in_parola = 0;
14    char c;
15
16    while ((c = fgetc(file)) != EOF) {
17        caratteri++;
18
19        if (c == '\n') {
20            righe++;
21        }
22
23        if (isspace(c)) {
24            in_parola = 0;
25        } else if (!in_parola) {
26            in_parola = 1;
27            parole++;
28        }
29    }
30
31    printf("Statistiche del file:\n");
32    printf("Righe: %d\n", righe);
33    printf("Parole: %d\n", parole);
34    printf("Caratteri: %d\n", caratteri);
35
36    fclose(file);
37    return 0;
38 }
```

### 10.8.3 Database Studenti su File

```
1  #include <stdio.h>
2  #include <string.h>
3
4  typedef struct {
5      int matricola;
6      char nome[30];
7      char cognome[30];
8      float media;
9  } Studente;
10
11 void aggiungi_studente() {
12     FILE *file = fopen("studenti.dat", "ab");
13     if (file == NULL) {
14         printf("Errore apertura file!\n");
15         return;
16     }
17
18     Studente s;
19     printf("Matricola: ");
20     scanf("%d", &s.matricola);
21     printf("Nome: ");
22     scanf("%s", s.nome);
23     printf("Cognome: ");
24     scanf("%s", s.cognome);
25     printf("Media: ");
26     scanf("%f", &s.media);
27
28     fwrite(&s, sizeof(Studente), 1, file);
29     printf("Studente aggiunto!\n");
30
31     fclose(file);
32 }
33
34 void visualizza_studenti() {
35     FILE *file = fopen("studenti.dat", "rb");
36     if (file == NULL) {
37         printf("Nessun dato disponibile!\n");
38         return;
39     }
40
41     Studente s;
42     printf("\n=== ELENCO STUDENTI ===\n");
43
44     while (fread(&s, sizeof(Studente), 1, file) == 1) {
45         printf("Matricola: %d\n", s.matricola);
46         printf("Nome: %s %s\n", s.nome, s.cognome);
47         printf("Media: %.2f\n", s.media);
48         printf("-----\n");
49     }
50
51     fclose(file);
52 }
53
```

```
54 void cerca_studente(int matricola) {
55     FILE *file = fopen("studenti.dat", "rb");
56     if (file == NULL) {
57         printf("Nessun dato disponibile!\n");
58         return;
59     }
60
61     Studente s;
62     int trovato = 0;
63
64     while (fread(&s, sizeof(Studente), 1, file) == 1) {
65         if (s.matricola == matricola) {
66             printf("\nStudente trovato:\n");
67             printf("Matricola: %d\n", s.matricola);
68             printf("Nome: %s %s\n", s.nome, s.cognome);
69             printf("Media: %.2f\n", s.media);
70             trovato = 1;
71             break;
72         }
73     }
74
75     if (!trovato) {
76         printf("Studente non trovato!\n");
77     }
78
79     fclose(file);
80 }
81
82 int main(int argc, char** argv) {
83     int scelta, matricola;
84
85     do {
86         printf("\n=== GESTIONE STUDENTI ===\n");
87         printf("1. Aggiungi studente\n");
88         printf("2. Visualizza tutti\n");
89         printf("3. Cerca studente\n");
90         printf("0. Esci\n");
91         printf("Scelta: ");
92         scanf("%d", &scelta);
93
94         switch (scelta) {
95             case 1:
96                 aggiungi_studente();
97                 break;
98             case 2:
99                 visualizza_studenti();
100                break;
101             case 3:
102                 printf("Inserisci matricola: ");
103                 scanf("%d", &matricola);
104                 cerca_studente(matricola);
105                 break;
106             case 0:
107                 printf("Arrivederci!\n");
108                 break;
109             default:
110                 printf("Scelta non valida!\n");
111         }
```

```
112     } while (scelta != 0);
113
114     return 0;
115 }
```

#### 10.8.4 File CSV

```
1  #include <stdio.h>
2  #include <string.h>
3
4  typedef struct {
5      char nome[30];
6      char cognome[30];
7      int eta;
8      char citta[30];
9  } Persona;
10
11 void scrivi_csv() {
12     FILE *file = fopen("persone.csv", "w");
13     if (file == NULL) {
14         printf("Errore!\n");
15         return;
16     }
17
18     // Intestazione
19     fprintf(file, "Nome,Cognome,Eta,Citta\n");
20
21     // Dati
22     fprintf(file, "Mario,Rossi,30,Milano\n");
23     fprintf(file, "Lucia,Bianchi,25,Roma\n");
24     fprintf(file, "Giovanni,Verdi,35,Napoli\n");
25
26     fclose(file);
27     printf("File CSV creato!\n");
28 }
29
30 void leggi_csv() {
31     FILE *file = fopen("persone.csv", "r");
32     if (file == NULL) {
33         printf("File non trovato!\n");
34         return;
35     }
36
37     char linea[200];
38     Persona p;
39
40     // Salta l'intestazione
41     fgets(linea, sizeof(linea), file);
42
43     printf("\n=== DATI DA CSV ===\n");
44
45     while (fgets(linea, sizeof(linea), file) != NULL) {
46         // Parse della riga CSV
47         sscanf(linea, "%[^,],%[^,],%d,%[^\\n]",
48             p.nome, p.cognome, &p.eta, p.citta);
49
50         printf("%s %s, %d anni, %s\n",
```

```
51         p.nome, p.cognome, p.eta, p.citta);
52     }
53
54     fclose(file);
55 }
56
57 int main(int argc, char** argv) {
58     scrivi_csv();
59     leggi_csv();
60     return 0;
61 }
```

## 10.9 Gestione Errori

### 10.9.1 perror

Stampa un messaggio di errore descrittivo.

```
1  #include <stdio.h>
2  #include <errno.h>
3
4  int main(int argc, char** argv) {
5      FILE *file = fopen("nonexistent.txt", "r");
6
7      if (file == NULL) {
8          perror("Errore"); // Stampa: "Errore: No such file or
9                             directory"
10         return 1;
11     }
12
13     fclose(file);
14     return 0;
15 }
```

## 10.10 Esercizi

### 10.10.1 Livello Base

1. Scrivi un programma che crea un file di testo e ci scrive 10 numeri casuali.
2. Crea un programma che legge un file di testo e stampa solo le righe che contengono una parola specifica.
3. Implementa un programma che conta quante volte appare un carattere in un file.
4. Scrivi un programma che unisce due file di testo in un terzo file.

### 10.10.2 Livello Intermedio

1. Crea un programma che ordina le righe di un file in ordine alfabetico.
2. Implementa un sistema di log che registra data, ora e messaggio su file.

3. Scrivi un programma che cifra e decifra un file di testo con cifrario di Cesare.
4. Crea un programma che gestisce una rubrica telefonica salvata su file CSV.

### 10.10.3 Livello Avanzato

1. Implementa un editor di testo semplice che permette di modificare righe specifiche di un file.
2. Crea un sistema di gestione inventario con salvataggio su file binario e funzioni di ricerca.
3. Scrivi un programma che comprime un file di testo eliminando spazi multipli e righe vuote.
4. Implementa un sistema di backup incrementale che salva solo le modifiche rispetto all'ultima versione.

# Capitolo 11

## Esercizi Completi

Questo capitolo raccoglie tutti gli esercizi presenti nei capitoli precedenti, organizzati per argomento e livello di difficoltà.

### 11.1 Variabili e Tipi di Dati

#### 11.1.1 Livello Base

1. Scrivi un programma che dichiara variabili di tutti i tipi base e stampa i loro valori.
2. Crea un programma che calcola l'area di un cerchio dato il raggio.
3. Implementa un programma che converte euro in dollari (tasso fisso 1.2).
4. Scrivi un programma che calcola la media di tre numeri float inseriti dall'utente.

#### 11.1.2 Livello Intermedio

1. Crea un programma che converte un numero di secondi in ore, minuti e secondi.
2. Implementa un convertitore tra diverse unità di temperatura (Celsius, Fahrenheit, Kelvin).
3. Scrivi un programma che calcola l'IMC (Indice di Massa Corporea) e fornisce una valutazione.
4. Crea un programma che calcola interessi composti dato capitale, tasso e periodo.

#### 11.1.3 Livello Avanzato

1. Implementa un programma che risolve equazioni di secondo grado complete.
2. Crea un convertitore universale di unità di misura (lunghezza, peso, volume, temperatura).
3. Scrivi un programma che calcola la distanza tra due coordinate GPS.
4. Implementa un sistema di calcolo del codice fiscale dato nome, cognome, data e luogo di nascita.

## 11.2 Operatori ed Espressioni

### 11.2.1 Livello Base

1. Scrivi un programma che calcola l'area e il perimetro di un rettangolo dati base e altezza.
2. Crea un programma che converte una temperatura da Celsius a Fahrenheit.
3. Scrivi un programma che verifica se un numero è pari o dispari.
4. Implementa un programma che calcola la media di tre numeri float.

### 11.2.2 Livello Intermedio

1. Scrivi un programma che dato un numero di secondi, calcola quante ore, minuti e secondi rappresenta.
2. Crea un programma che scambia i valori di due variabili senza usare una variabile temporanea.
3. Implementa un programma che verifica se un anno è bisestile.
4. Scrivi un programma che estrae le singole cifre di un numero intero a tre cifre.

### 11.2.3 Livello Avanzato

1. Crea un programma che implementa una calcolatrice semplice con le quattro operazioni base.
2. Scrivi un programma che usa operatori bit a bit per verificare se un numero è una potenza di 2.
3. Implementa l'algoritmo di Euclide per il calcolo del MCD.
4. Crea un programma che conta i bit impostati a 1 nella rappresentazione binaria di un numero.

## 11.3 Controllo di Flusso

### 11.3.1 Livello Base

1. Scrivi un programma che determina se un numero è positivo, negativo o zero.
2. Crea un programma che stampa i numeri da 1 a 100.
3. Scrivi un programma che calcola la somma dei primi N numeri naturali.
4. Implementa la tavola pitagorica di un numero dato.

### 11.3.2 Livello Intermedio

1. Scrivi un programma che verifica se un numero è primo.
2. Crea un programma che inverte un numero (es: 1234 diventa 4321).
3. Implementa l'algoritmo di Euclide per il MCD.
4. Disegna un triangolo di numeri crescenti.

### 11.3.3 Livello Avanzato

1. Genera la sequenza di Fibonacci fino a N termini.
2. Verifica se un numero è palindromo.
3. Trova tutti i numeri perfetti minori di 1000.
4. Disegna il triangolo di Floyd.

## 11.4 Funzioni

### 11.4.1 Livello Base

1. Scrivi una funzione che converte temperatura da Celsius a Fahrenheit.
2. Crea una funzione che verifica se un numero è pari.
3. Implementa una funzione che calcola l'area di un cerchio dato il raggio.
4. Scrivi una funzione che stampa un triangolo di asterischi di altezza N.

### 11.4.2 Livello Intermedio

1. Crea una funzione che inverte un numero intero.
2. Scrivi una funzione ricorsiva che calcola la somma delle cifre di un numero.
3. Implementa una funzione che verifica se una stringa è palindroma.
4. Crea funzioni per conversione tra unità di misura.

### 11.4.3 Livello Avanzato

1. Implementa le Torre di Hanoi in modo ricorsivo.
2. Genera numeri primi con il Crivello di Eratostene.
3. Calcola ricorsivamente il coefficiente binomiale.
4. Risolvi il problema delle N regine con backtracking.

## 11.5 Array

### 11.5.1 Livello Base

1. Leggi 10 numeri e stampa quanti sono pari e quanti dispari.
2. Trova la seconda cifra più grande in un array.
3. Conta le occorrenze di un numero in un array.
4. Verifica se un array è palindromo.

### 11.5.2 Livello Intermedio

1. Implementa la ricerca binaria su array ordinato.
2. Ruota gli elementi di un array di K posizioni.
3. Fondi due array ordinati in un unico array ordinato.
4. Trova sottoinsiemi con somma uguale a un valore dato.

### 11.5.3 Livello Avanzato

1. Implementa Quick Sort ricorsivo.
2. Moltiplica due matrici.
3. Trova la sotto-sequenza contigua con somma massima (Kadane).
4. Risolvi il problema dello zaino 0/1 con programmazione dinamica.

## 11.6 Puntatori

### 11.6.1 Livello Base

1. Usa puntatori per scambiare tre valori ciclicamente.
2. Inverti un array usando puntatori.
3. Trova max e min restituendoli tramite puntatori.
4. Alloca dinamicamente un array con i quadrati dei numeri da 1 a N.

### 11.6.2 Livello Intermedio

1. Concatena due array allocati dinamicamente.
2. Rimuovi duplicati da array dinamico.
3. Implementa matrice dinamica (array 2D).
4. Ordina usando solo aritmetica dei puntatori.

### 11.6.3 Livello Avanzato

1. Implementa lista collegata con inserimento, ricerca e cancellazione.
2. Crea uno stack dinamico.
3. Gestisci matrice sparsa con allocazione dinamica.
4. Implementa albero binario di ricerca.

## 11.7 Stringhe

### 11.7.1 Livello Base

1. Conta le vocali in una stringa.
2. Verifica se una stringa contiene solo caratteri alfabetici.
3. Rimuovi caratteri non alfanumerici da stringa.
4. Stampa ogni parola di una frase su una riga separata.

### 11.7.2 Livello Intermedio

1. Sostituisci tutte le occorrenze di una sottostringa.
2. Verifica se due stringhe sono anagrammi.
3. Ordina array di stringhe in ordine alfabetico.
4. Comprimi stringa (es: "aaabbc"  $\rightarrow$  "a3b2c1").

### 11.7.3 Livello Avanzato

1. Parser di espressioni matematiche semplici.
2. Valida indirizzi email.
3. Pattern matching con wildcard \* e ?.
4. Sistema di cifratura/decifratura Caesar.

## 11.8 Struct

### 11.8.1 Livello Base

1. Struct Rettangolo con funzioni per area e perimetro.
2. Struct Tempo con conversione in secondi totali.
3. Struct ContoCorrente con operazioni deposito/prelievo.
4. Struct Prodotto per gestire inventario.

### 11.8.2 Livello Intermedio

1. Sistema rubrica telefonica con array di struct.
2. Struct Matrice2D con allocazione dinamica.
3. Database libri con ricerca e ordinamento.
4. Struct per poligoni con calcolo perimetro e area.

### 11.8.3 Livello Avanzato

1. Lista collegata con struct e puntatori.
2. Sistema gestione parcheggio.
3. Implementa grafo con struct per nodi e archi.
4. Sistema bancario con conti, transazioni e storici.

## 11.9 File

### 11.9.1 Livello Base

1. Crea file con 10 numeri casuali.
2. Leggi file e stampa righe contenenti parola specifica.
3. Conta occorrenze di un carattere in file.
4. Unisci due file in un terzo file.

### 11.9.2 Livello Intermedio

1. Ordina righe di file in ordine alfabetico.
2. Sistema di log con data, ora e messaggio.
3. Cifra e decifra file con cifrario Caesar.
4. Rubrica telefonica salvata su CSV.

### 11.9.3 Livello Avanzato

1. Editor di testo per modificare righe specifiche.
2. Sistema inventario con file binario e ricerca.
3. Comprimi file eliminando spazi multipli e righe vuote.
4. Backup incrementale con salvataggio modifiche.

## 11.10 Progetti Completi

### 11.10.1 Progetto 1: Sistema di Gestione Biblioteca

Crea un sistema completo per gestire una biblioteca con:

- Struct per Libro (titolo, autore, ISBN, disponibilità)
- Struct per Utente (nome, tessera, libri presi in prestito)
- Funzioni per aggiungere/rimuovere libri
- Funzioni per prestito e restituzione
- Ricerca libri per autore/titolo
- Salvataggio su file

### 11.10.2 Progetto 2: Gioco del Tris

Implementa il gioco del tris con:

- Matrice 3x3 per la griglia
- Funzione per stampare la griglia
- Funzione per verificare vittoria/pareggio
- Modalità 2 giocatori
- Validazione input
- Possibilità di giocare più partite

### 11.10.3 Progetto 3: Gestionale Studenti

Sistema completo per gestire studenti con:

- Struct Studente con dati personali e voti
- Array dinamico di studenti
- Menu interattivo
- Funzioni CRUD (Create, Read, Update, Delete)
- Calcolo medie e statistiche
- Ordinamento per voto/nome
- Salvataggio su file binario

#### 11.10.4 Progetto 4: Calcolatrice Avanzata

Calcolatrice con funzionalità avanzate:

- Operazioni base (+, -, \*, /)
- Potenze e radici
- Funzioni trigonometriche
- Conversioni tra basi numeriche
- Storico operazioni
- Salvataggio storico su file

#### 11.10.5 Progetto 5: Sistema di Prenotazioni

Sistema per gestire prenotazioni (es: ristorante, hotel):

- Struct per Prenotazione
- Gestione disponibilità
- Ricerca per data/nome
- Cancellazione prenotazioni
- Report statistici
- Salvataggio persistente

### 11.11 Esercizi di Riepilogo

#### 11.11.1 Esercizio Complessivo 1: Gestione Magazzino

Crea un programma completo che gestisce un magazzino con:

- Prodotti (codice, nome, quantità, prezzo)
- Carico e scarico merci
- Inventario con valore totale
- Ricerca prodotti sotto scorta
- Report su file

### 11.11.2 Esercizio Complessivo 2: Agenda Personale

Implementa un'agenda con:

- Eventi con data, ora, descrizione
- Visualizzazione giornaliera/settimanale
- Ricerca eventi
- Promemoria
- Esportazione in formato testo

### 11.11.3 Esercizio Complessivo 3: Analizzatore di Testo

Programma che analizza file di testo fornendo:

- Numero righe, parole, caratteri
- Parole più frequenti
- Lunghezza media parole
- Indice di leggibilità
- Report completo su file

## 11.12 Consigli per gli Esercizi

### 11.12.1 Come Affrontare gli Esercizi

1. **Leggi attentamente:** comprendi cosa ti viene chiesto
2. **Scomponi il problema:** dividi in sotto-problemi più semplici
3. **Pianifica:** pensa alla struttura prima di scrivere codice
4. **Inizia dal semplice:** parti da una versione base e migliora
5. **Testa frequentemente:** verifica il codice passo dopo passo
6. **Gestisci gli errori:** prevedi input non validi
7. **Commenta il codice:** spiega la logica complessa
8. **Refactoring:** migliora il codice dopo che funziona

### 11.12.2 Debugging

Quando il programma non funziona:

1. Usa `printf()` per stampare valori intermedi
2. Verifica che gli indici degli array siano corretti

3. Controlla i puntatori (NULL, inizializzazione)
4. Verifica apertura/chiusura file
5. Controlla le condizioni nei cicli e negli if
6. Usa il debugger (gdb su Linux/Mac, Visual Studio su Windows)

### 11.12.3 Best Practices

- Usa nomi di variabili descrittivi
- Mantieni funzioni piccole e focalizzate
- Evita duplicazione di codice
- Gestisci sempre gli errori
- Libera sempre la memoria allocata
- Chiudi sempre i file aperti
- Commenta il codice complesso
- Usa costanti invece di "numeri magici"
- Testa casi limite (0, negativi, stringhe vuote)

## 11.13 Risorse Aggiuntive

### 11.13.1 Compilazione

Comandi base per compilare programmi C:

```
# Compilazione semplice
gcc programma.c -o programma

# Con warning
gcc -Wall -Wextra programma.c -o programma

# Con debugging
gcc -g programma.c -o programma

# Con ottimizzazione
gcc -O2 programma.c -o programma

# Con libreria matematica
gcc programma.c -o programma -lm
```

### 11.13.2 Online Judge per Esercitarsi

- [HackerRank](#)
- [LeetCode](#)
- [Codeforces](#)
- [Project Euler](#)
- [Codewars](#)

### 11.13.3 Documenta zione di Riferimento

- [cppreference.com](#)
- [The C Programming Language \(K&R\)](#)
- [man pages \(Linux/Mac\)](#)
- [MSDN \(Windows\)](#)

## Capitolo 12

# Makefile e Build Automation

### 12.1 Introduzione ai Makefile

Un Makefile è uno strumento che automatizza il processo di compilazione e costruzione (build) di un progetto C. Permette di:

- Automatizzare comandi di compilazione complessi
- Gestire dipendenze tra file oggetto e sorgenti
- Compilare solo i file modificati
- Eseguire test e pulizia automaticamente
- Rendere il progetto portabile e facile da distribuire

#### Nota

Un Makefile usa il comando **make** per eseguire le regole definite. La sintassi è sensibile alle tabulazioni (tab, non spazi).

#### 12.1.1 Obiettivi di apprendimento

Al termine di questo capitolo sarai in grado di:

- Comprendere la struttura di un Makefile
- Definire variabili e regole di compilazione
- Creare target per compilazione, pulizia e test
- Gestire dipendenze tra file
- Ottimizzare il processo di build
- Scrivere Makefile portatili e robusti

## 12.2 Struttura Base di un Makefile

### 12.2.1 Sintassi di una Regola

```
1 target: dipendenze
2     comando1
3     comando2
```

Una regola Makefile contiene:

1. **target**: il nome del file o dell'azione da creare
2. **dipendenze**: file da cui dipende il target
3. **comandi**: comandi da eseguire (con tabulazione iniziale)

### 12.2.2 Esempio Semplice

```
1 # Makefile semplice per compilare un programma C
2
3 # Compilazione del programma principale
4 programma: main.o funzioni.o
5     gcc -o programma main.o funzioni.o
6
7 # Compilazione di main.c in main.o
8 main.o: main.c
9     gcc -c main.c
10
11 # Compilazione di funzioni.c in funzioni.o
12 funzioni.o: funzioni.c
13     gcc -c funzioni.c
14
15 # Cancella file oggetto e eseguibile
16 clean:
17     rm -f programma main.o funzioni.o
```

### 12.2.3 Esecuzione

```
1 # Compila il target 'programma'
2 make programma
3
4 # Se 'programma' è il primo target, basta scrivere:
5 make
6
7 # Pulisce i file compilati
8 make clean
```

## 12.3 Variabili e Pattern

### 12.3.1 Variabili Comuni

Le variabili permettono di evitare ripetizioni e rendere il Makefile manutenibile.

```

1 # Dichiarazione di variabili
2 CC = gcc
3 CFLAGS = -Wall -Wextra -std=c99
4 LDFLAGS = -lm
5
6 # Variabili automatiche
7 # $@ = nome del target
8 # $< = primo file di dipendenza
9 # $^ = tutti i file di dipendenza
10 # $* = radice del nome del target
11
12 # Utilizzo delle variabili
13 main.o: main.c
14     $(CC) $(CFLAGS) -c $< -o $@
15
16 programma: main.o funzioni.o
17     $(CC) -o $@ $^ $(LDFLAGS)

```

### 12.3.2 Pattern Rules

Le pattern rule definiscono come compilare categorie di file.

```

1 # Pattern rule: compila tutti i file .c in file .o
2 %.o: %.c
3     $(CC) $(CFLAGS) -c $< -o $@
4
5 # Pattern rule: compila tutti i file .c in eseguibili
6 %: %.c
7     $(CC) $(CFLAGS) $< -o $@ $(LDFLAGS)

```

### 12.3.3 Variabili Integrate

```

1 # Variabili di sistema predefinite in make
2
3 CC = gcc                # Compilatore C
4 CFLAGS = -Wall          # Flag di compilazione
5 LDFLAGS = -lm           # Flag di linking
6 RM = rm -f              # Comando per rimuovere
7 SOURCES = $(wildcard *.c) # Espande a tutti i file .c
8 OBJECTS = $(SOURCES:.c=.o) # Sostituisce .c con .o
9
10 # Utilizzo
11 programma: $(OBJECTS)
12     $(CC) -o programma $^ $(LDFLAGS)

```

#### Attenzione

Ricorda: i comandi dentro le regole devono iniziare con una tabulazione (tab), non con spazi. Questo è un errore comune che causa problemi.

## 12.4 Makefile Completo con Esercizi

### 12.4.1 Progetto Multi-File

```

1  # =====
2  # Makefile per un progetto C multi-file
3  # =====
4
5  # Variabili di compilazione
6  CC = gcc
7  CFLAGS = -Wall -Wextra -std=c99 -g
8  LDFLAGS = -lm
9
10 # Nome dell'eseguibile
11 EXEC = programma
12
13 # File sorgenti e oggetto
14 SOURCES = main.c funzioni.c utilita.c
15 HEADERS = funzioni.h utilita.h
16 OBJECTS = $(SOURCES:.c=.o)
17
18 # =====
19 # TARGET PRINCIPALE
20 # =====
21
22 # Target di default: compilazione
23 all: $(EXEC)
24
25 # Creazione dell'eseguibile
26 $(EXEC): $(OBJECTS)
27     $(CC) -o $@ $^ $(LDFLAGS)
28     @echo "Compilazione completata: $(EXEC)"
29
30 # Pattern rule per compilare .c in .o
31 %.o: %.c $(HEADERS)
32     $(CC) $(CFLAGS) -c $< -o $@
33
34 # =====
35 # TARGET AUSILIARIO
36 # =====
37
38 # Pulizia dei file temporanei
39 clean:
40     $(RM) $(OBJECTS)
41     @echo "File oggetto rimossi"
42
43 # Pulizia totale
44 distclean: clean
45     $(RM) $(EXEC)
46     @echo "Eseguibile rimosso"
47
48 # Ricompilazione completa
49 rebuild: distclean all
50
51 # =====
52 # TARGET DI DEBUG E TEST
53 # =====
54

```

```

55 # Compilazione con flag di debug
56 debug: CFLAGS += -O0 -g3 -DDEBUG
57 debug: clean $(EXEC)
58     @echo "Build di debug completata"
59
60 # Test eseguendo il programma
61 test: $(EXEC)
62     ./$$(EXEC)
63
64 # =====
65 # TARGET FITTIZI (non corrispondono a file)
66 # =====
67
68 .PHONY: all clean distclean rebuild debug test

```

### 12.4.2 Progetto con Sottodirectory

```

1 # Makefile per progetto con organizzazione in directory
2
3 SRC_DIR = src
4 OBJ_DIR = obj
5 BIN_DIR = bin
6 INC_DIR = include
7
8 CC = gcc
9 CFLAGS = -Wall -Wextra -std=c99 -I$(INC_DIR)
10 LDFLAGS = -lm
11
12 EXEC = $(BIN_DIR)/programma
13
14 # Trova tutti i file .c nella directory src
15 SOURCES = $(wildcard $(SRC_DIR)/*.c)
16 OBJECTS = $(SOURCES:$(SRC_DIR)/%.c=$(OBJ_DIR)/%.o)
17
18 all: $(EXEC)
19
20 # Crea directory se non esistono
21 $(OBJ_DIR):
22     mkdir -p $(OBJ_DIR)
23
24 $(BIN_DIR):
25     mkdir -p $(BIN_DIR)
26
27 # Compilazione dell'eseguibile
28 $(EXEC): $(OBJECTS) | $(BIN_DIR)
29     $(CC) -o $@ $^ $(LDFLAGS)
30     @echo "Eseguibile creato: $@"
31
32 # Compilazione degli oggetti
33 $(OBJ_DIR)/%.o: $(SRC_DIR)/%.c | $(OBJ_DIR)
34     $(CC) $(CFLAGS) -c $< -o $@
35
36 clean:
37     rm -rf $(OBJ_DIR) $(BIN_DIR)
38     @echo "Pulito"
39
40 .PHONY: all clean

```

### 12.4.3 Makefile Avanzato con Documentazione

```

1  # =====
2  # MAKEFILE AVANZATO - PROGETTO C
3  # =====
4  # Utilizzo: make [target]
5  # Targets disponibili:
6  #   all      - compila il programma (default)
7  #   clean    - rimuove file oggetto
8  #   distclean - rimuove tutto
9  #   rebuild  - ricompila da zero
10 #   run      - esegue il programma
11 #   help     - mostra questo messaggio
12
13 CC = gcc
14 CFLAGS = -Wall -Wextra -std=c99 -D_DEFAULT_SOURCE
15 LDFLAGS = -lm
16 DEBUGFLAGS = -g3 -O0
17
18 EXEC = prog
19 SOURCES = $(wildcard *.c)
20 OBJECTS = $(SOURCES:.c=.o)
21
22 # Verificazione di make version
23 ifeq ($(shell make --version | head -1 | grep -o GNU),GNU)
24     SHELL := /bin/bash
25 endif
26
27 all: $(EXEC)
28
29 $(EXEC): $(OBJECTS)
30     @echo "[LINKING] $@"
31     $(CC) -o $@ $^ $(LDFLAGS)
32
33 %.o: %.c
34     @echo "[COMPILE] $<"
35     $(CC) $(CFLAGS) -c $< -o $@
36
37 clean:
38     @echo "[CLEAN] Rimozione file oggetto"
39     $(RM) $(OBJECTS)
40
41 distclean: clean
42     @echo "[DISTCLEAN] Rimozione eseguibile"
43     $(RM) $(EXEC)
44
45 rebuild: distclean all
46
47 run: $(EXEC)
48     @echo "[RUN] Esecuzione di $(EXEC)"
49     ./$(EXEC)
50
51 debug: CFLAGS += $(DEBUGFLAGS)
52 debug: clean $(EXEC)
53

```

```

54 help:
55     @echo "Targets disponibili:"
56     @echo "  all      - Compilazione (default)"
57     @echo "  clean    - Rimozione file oggetto"
58     @echo "  distclean - Rimozione tutto"
59     @echo "  rebuild  - Ricompilazione da zero"
60     @echo "  run      - Esecuzione del programma"
61     @echo "  debug    - Compilazione con debug"
62     @echo "  help     - Questo messaggio"
63
64 .PHONY: all clean distclean rebuild run debug help

```

### Attenzione

Quando usare un Makefile:

- Progetti con più file sorgenti
- Compilazioni frequenti durante lo sviluppo
- Automazione di task ripetitivi
- Progetti che altri sviluppatori useranno

## 12.5 Dipendenze e Ottimizzazione

### 12.5.1 File di Dipendenza Automatiche

```

1  # Generazione automatica delle dipendenze include
2
3  CC = gcc
4  CFLAGS = -Wall -Wextra -std=c99
5  LDFLAGS = -lm
6
7  EXEC = programma
8  SOURCES = main.c funzioni.c
9  OBJECTS = $(SOURCES:.c=.o)
10 DEPENDS = $(SOURCES:.c=.d)
11
12 all: $(EXEC)
13
14 $(EXEC): $(OBJECTS)
15     $(CC) -o $@ $^ $(LDFLAGS)
16
17 %.o: %.c
18     $(CC) $(CFLAGS) -c $< -o $@
19     $(CC) -MM $< > $*.d
20
21 -include $(DEPENDS)
22
23 clean:
24     rm -f $(OBJECTS) $(DEPENDS) $(EXEC)
25
26 .PHONY: all clean

```

## 12.5.2 Compilazione Parallela

```
1 # Sfrutta processori multi-core
2
3 # Esegui con: make -j 4
4 # (-j 4 significa 4 processi paralleli)
5
6 CC = gcc
7 CFLAGS = -Wall -Wextra -std=c99
8 EXEC = programma
9 SOURCES = $(wildcard *.c)
10 OBJECTS = $(SOURCES:.c=.o)
11
12 all: $(EXEC)
13
14 $(EXEC): $(OBJECTS)
15     $(CC) -o $@ $^
16
17 %.o: %.c
18     $(CC) $(CFLAGS) -c $< -o $@
19
20 clean:
21     $(RM) $(OBJECTS) $(EXEC)
22
23 .PHONY: all clean
24
25 # Da terminale: make -j
```

## 12.6 Esercizi

### 12.6.1 Livello Base

1. Crea un Makefile per un progetto con due file: `main.c` e `calcoli.c`. Il Makefile deve avere target `all`, `clean`.
2. Scrivi un Makefile che compila un singolo file `hello.c` con flag `-Wall -Wextra`. Aggiungi un target `run` che esegue il programma.
3. Crea un Makefile per compilare tre file: `main.c`, `lettore.c`, `stampante.c`. Usa variabili per il compilatore e i flag.
4. Scrivi un Makefile che include pattern rules per compilare automaticamente qualsiasi file `.c` in `.o`.

### 12.6.2 Livello Intermedio

1. Crea un Makefile per un progetto con directory separate: `src/`, `obj/`, `bin/`. Il compilato deve essere in `bin/` e gli oggetti in `obj/`.
2. Implementa un Makefile con target aggiuntivi: `debug` (compila con `-g`), `release` (con `-O2`), `profile` (con `-pg`).
3. Scrivi un Makefile che genera automaticamente le dipendenze include tra file con il comando `gcc -MM`.

4. Crea un Makefile che visualizza messaggi informativi durante la compilazione (es: "[COMPILE] main.c", "[LINKING] programma").

### 12.6.3 Livello Avanzato

1. Implementa un Makefile che supporta compilazione parallela e misura il tempo di compilazione con `time make`.
2. Crea un sistema di Makefile gerarchico: un Makefile principale che chiama Makefile in sottodirectory.
3. Scrivi un Makefile che genera documentazione con Doxygen, esegue valgrind per memory leak detection, e crea un archivio tar del sorgente.
4. Implementa un Makefile che gestisce diverse configurazioni di build (debug, release, profiling) con dipendenze condizionali.

## 12.7 Riepilogo

- Un Makefile automatizza la compilazione e la costruzione di progetti C
- Le regole hanno la forma: `target: dipendenze` seguita da comandi con tabulazione
- Le variabili (come `CC`, `CFLAGS`) rendono il Makefile manutenibile
- Le pattern rules (`%.o: %.c`) evitano ripetizioni
- I target fittizi (`.PHONY`) eseguono azioni senza creare file
- `make clean` rimuove file compilati, `make distclean` tutto
- I comandi nelle regole DEVONO iniziare con tabulazione, non spazi
- Usa `make -j` per compilazione parallela

# Capitolo 13

## GDB Debugger

### 13.1 Introduzione al Debugging

Il debugging è il processo di ricerca e correzione degli errori (bug) in un programma. GDB (GNU Debugger) è uno strumento potente che permette di:

- Eseguire il programma passo dopo passo
- Impostare breakpoint per fermare l'esecuzione in punti specifici
- Ispezionare il valore delle variabili durante l'esecuzione
- Visualizzare lo stack di chiamate (call stack)
- Modificare il comportamento del programma al volo
- Rilevare errori di memoria e segmentation fault

#### Nota

Per usare GDB è necessario compilare il programma con il flag `-g` che include informazioni di debugging nel binario.

#### 13.1.1 Obiettivi di apprendimento

Al termine di questo capitolo sarai in grado di:

- Compilare programmi con informazioni di debug
- Avviare GDB e navigare il programma
- Usare breakpoint e watchpoint
- Ispezionare variabili e memoria
- Visualizzare e navigare lo stack
- Risolvere errori comuni nei programmi C
- Eseguire debugging su crash e segmentation fault

## 13.2 Compilazione per il Debugging

### 13.2.1 Flag -g

```
1 # Compilazione normale (senza debug)
2 gcc -o programma main.c
3
4 # Compilazione con debug (include simboli)
5 gcc -g -o programma main.c
6
7 # Compilazione con debug completo
8 gcc -g -O0 -o programma main.c
9
10 # Compilazione con debug e warning
11 gcc -g -Wall -Wextra -o programma main.c
```

Il flag -g include informazioni di debugging:

- Nomi di funzioni e variabili
- Numeri di linea del sorgente
- Tipi di dati
- Path ai file sorgenti

### 13.2.2 Esempio Programma con Bug

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void stampa_caratteri(char* stringa) {
5     // Bug: accesso oltre il limite dell'array
6     for (int i = 0; i <= 10; i++) {
7         printf("Carattere %d: %c\n", i, stringa[i]);
8     }
9 }
10
11 int main(int argc, char** argv) {
12     char messaggio[] = "Ciao";
13
14     printf("Lunghezza: %lu\n", strlen(messaggio));
15     stampa_caratteri(messaggio);
16
17     return 0;
18 }
```

#### Attenzione

Questo programma ha un buffer overflow. Non fare mai:

- `for (int i = 0; i <= dimensione; i++)` - dev'essere `i < dimensione`
- Accessi senza verificare i limiti dell'array
- Operazioni su puntatori NULL

## 13.3 Comandi Fondamentali di GDB

### 13.3.1 Avviare GDB

```
1 # Avvia GDB con il programma
2 gdb ./programma
3
4 # GDB prompt (gdb)
5
6 # Esci da GDB
7 quit
```

### 13.3.2 Breakpoint

```
1 # Imposta breakpoint sulla linea 10 del file main.c
2 break main.c:10
3 b main.c:10
4
5 # Imposta breakpoint su una funzione
6 break stampa_caratteri
7 b stampa_caratteri
8
9 # Imposta breakpoint con condizione
10 break main.c:20 if i > 5
11
12 # Visualizza tutti i breakpoint
13 info break
14
15 # Cancella un breakpoint
16 delete 1
17 del 1
18
19 # Disabilita/abilita un breakpoint senza cancellarlo
20 disable 1
21 enable 1
```

### 13.3.3 Esecuzione del Programma

```
1 # Esegue il programma fino al primo breakpoint (o fine)
2 run
3
4 # Esegue con argomenti da linea di comando
5 run argomento1 argomento2
6
7 # Continua l'esecuzione dal breakpoint corrente
8 continue
9 c
10
11 # Esegue la prossima linea (salta dentro le funzioni)
12 next
13 n
14
15 # Esegue la prossima linea (entra dentro le funzioni)
16 step
```

```
17 s
18
19 # Continua finche' non torna dalla funzione attuale
20 finish
```

### 13.3.4 Ispezionare Variabili

```
1 # Stampa il valore di una variabile
2 print variabile
3 p variabile
4
5 # Stampa con formato specifico
6 p /x variabile      # Esadecimale
7 p /o variabile      # Ottale
8 p /t variabile      # Binario
9 p /d variabile      # Decimale
10
11 # Stampa un array/stringa
12 p array@10          # Stampa 10 elementi
13 p *puntatore
14
15 # Visualizza tipo di una variabile
16 whatis variabile
17
18 # Stampa più variabili locali
19 info locals
20
21 # Stampa le variabili globali
22 info globals
```

### 13.3.5 Watchpoint

```
1 # Imposta un watchpoint su una variabile
2 watch variabile
3
4 # Il programma si ferma quando la variabile cambia valore
5
6 # Visualizza watchpoint
7 info watch
8
9 # Cancella un watchpoint
10 delete numero_watchpoint
```

## 13.4 Stack e Funzioni

### 13.4.1 Stack Trace

```
1 # Visualizza lo stack di chiamate (call stack)
2 backtrace
3 bt
4
5 # Visualizza informazioni dettagliate dello stack
```

```
6 backtrace full
7
8 # Visualizza il numero di frame
9 frame
10
11 # Cambia frame (riga di livello nello stack)
12 frame 0
13 frame 1
14 frame 2
15
16 # Sali/scendi nello stack
17 up
18 down
```

### 13.4.2 Esempio di Debugging con Stack

```
1 #include <stdio.h>
2
3 void funzione_c() {
4     printf("In funzione_c\n");
5     int errore = 1 / 0; // Errore di divisione per zero
6 }
7
8 void funzione_b() {
9     printf("In funzione_b\n");
10    funzione_c();
11 }
12
13 void funzione_a() {
14     printf("In funzione_a\n");
15     funzione_b();
16 }
17
18 int main(int argc, char** argv) {
19     printf("In main\n");
20     funzione_a();
21     return 0;
22 }
```

Nel debugger, dopo il crash, digitare `backtrace` mostra la sequenza di chiamate.

## 13.5 Comandi Avanzati

### 13.5.1 Memoria e Puntatori

```
1 # Esamina memoria in un indirizzo
2 x 0x7fff5fbff8c0
3
4 # Formati differenti
5 x /x indirizzo      # Esadecimale
6 x /d indirizzo      # Intero con segno
7 x /s indirizzo      # Stringa
8 x /i indirizzo      # Istruzione assembler
9
```

```
10 # Mostra indirizzo di una variabile
11 p &variabile
12
13 # Dereferenzia un puntatore
14 p *puntatore
15
16 # Numero di elementi da mostrare
17 x /10x indirizzo      # Mostra 10 parole in hex
```

### 13.5.2 Modificare Valori

```
1 # Cambia il valore di una variabile durante l'esecuzione
2 set variable variabile = nuovo_valore
3 set var x = 42
4
5 # Cambia il valore di un registro
6 set $rax = 10
7
8 # Torna a un precedente stato? NO, ma puoi modificare
```

### 13.5.3 Script e Automazione

```
1 # File: debug_script.gdb
2
3 break main.c:20
4 run
5 print x
6 print y
7 print z
8 continue
9 quit
10
11 # Utilizzo:
12 # gdb programma -x debug_script.gdb
```

## 13.6 Esempi Pratici di Debugging

### 13.6.1 Debug di Segmentation Fault

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char** argv) {
6     char* buffer = NULL;
7
8     // BUG: accesso a puntatore NULL
9     strcpy(buffer, "Ciao");
10
11     printf("%s\n", buffer);
12
13     return 0;
```

```
14 }
```

Comandi GDB:

```
1 gdb ./programma
2 (gdb) run
3 # Segmentation fault (core dumped)
4 (gdb) backtrace
5 (gdb) frame 0
6 (gdb) print buffer
7 # $1 = 0x0 (NULL)
8 (gdb) quit
```

### 13.6.2 Debug di Buffer Overflow

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(int argc, char** argv) {
5     char buffer[10];
6
7     // BUG: scrive oltre i 10 byte
8     strcpy(buffer, "Questo è un messaggio molto lungo!");
9
10    printf("%s\n", buffer);
11
12    return 0;
13 }
```

Comandi GDB:

```
1 gdb ./programma
2 (gdb) break main
3 (gdb) run
4 (gdb) step
5 (gdb) print buffer
6 (gdb) x /s buffer
7 # Mostra il contenuto del buffer
```

### 13.6.3 Debug di Loop Infinito

```
1 #include <stdio.h>
2
3 int main(int argc, char** argv) {
4     int i = 0;
5
6     // BUG: loop infinito (i non cambia mai)
7     while (i < 10) {
8         printf("i = %d\n", i);
9         // Manca i++
10    }
11
12    return 0;
13 }
```

Comandi GDB:

```

1  gdb ./programma
2  (gdb) run
3  # Dopo un po', premi Ctrl+C per fermare il programma
4  (gdb) break (numero linea con printf)
5  (gdb) continue
6  (gdb) print i
7  # $1 = 0 (è sempre 0!)
8  (gdb) list
9  # Mostra il sorgente intorno alla posizione corrente

```

### Attenzione

Per fermare un programma in esecuzione in GDB, premi **Ctrl+C**. Questo non esce da GDB, ma ferma il programma, permettendoti di ispezionare lo stato corrente.

## 13.7 Comandi Utili Riassunti

### 13.7.1 Tabella Veloce

1	run / r	Esegui il programma
2	break / b	Imposta breakpoint
3	continue / c	Continua esecuzione
4	next / n	Linea successiva
5	step / s	Entra in funzione
6	finish / fin	Esci dalla funzione
7	backtrace / bt	Mostra call stack
8	print / p	Stampa variabile
9	info break	Lista breakpoint
10	info locals	Variabili locali
11	help comando	Aiuto su comando
12	quit / q	Esci da GDB
13	list / l	Mostra sorgente
14	watch	Monitora variabile
15	delete / del	Cancella breakpoint
16	disable / enable	Disabilita/abilita

## 13.8 Esercizi

### 13.8.1 Livello Base

1. Compila un programma semplice con `gcc -g` e avvialo in GDB. Usa il comando `run`, poi `quit`.
2. Scrivi un programma con un bug di accesso ad array fuori bounds. Debugga con GDB usando `break` e `print`.
3. Crea un programma con una funzione che modifica una variabile. Usa `watch` per monitorare quando cambia.
4. Scrivi un programma con un loop che stampa variabili. Usa `next` per eseguire linea per linea.

### 13.8.2 Livello Intermedio

1. Implementa un programma con 3 funzioni annidate. Debugga con `step` e `backtrace` per navigare le funzioni.
2. Crea un programma con un buffer overflow deliberato. Usa GDB per ispezionare la memoria corrotta.
3. Scrivi un programma che gestisce puntatori. Debugga con `print &variabile` e `x /x indirizzo`.
4. Implementa un programma con un segmentation fault. Usa GDB per trovare la linea del crash e il motivo.

### 13.8.3 Livello Avanzato

1. Crea un programma con un memory leak (malloc senza free). Usa valgrind con GDB per individuare il leak.
2. Implementa un debugger interattivo che permette di caricare un programma, impostare breakpoint, e ispezionare lo stato.
3. Scrivi uno script GDB (`.gdb`) che automatizza il debugging di un programma complesso con più breakpoint condizionali.
4. Crea un programma con race condition (multi-threading). Usa GDB thread debugging (`info threads`, `thread 1`).

## 13.9 Riepilogo

- GDB è il debugger ufficiale di GNU/Linux per programmi C/C++
- Compila con `gcc -g` per includere informazioni di debug
- `break` imposta breakpoint; `run` esegue; `continue` riprende
- `next` salta funzioni; `step` entra dentro; `finish` esce
- `print` mostra variabili; `watch` monitora cambiamenti
- `backtrace` mostra lo stack di chiamate
- `x` esamina memoria a indirizzi specifici
- I breakpoint condizionali (`if`) sono molto utili
- GDB permette di modificare variabili durante l'esecuzione con `set var`
- Script GDB (`.gdb`) automatizzano il debugging
- Combinato con valgrind per memory leak detection

# Capitolo 14

## Librerie Standard Avanzate

### 14.1 Introduzione alle Librerie Standard

Le librerie standard C forniscono funzioni predefinite per compiti comuni. Questo capitolo affronta le librerie più utili per:

- Manipolazione di stringhe con `string.h`
- Operazioni matematiche avanzate con `math.h`
- Gestione del tempo con `time.h`
- Allocazione e gestione della memoria con `stdlib.h`
- Ordinamento e ricerca di dati

#### Nota

Per usare la libreria matematica, compila con `gcc -lm` (opzione `-lm` aggiunge il linking alla libreria `math`).

#### 14.1.1 Obiettivi di apprendimento

Al termine di questo capitolo sarai in grado di:

- Usare le funzioni di manipolazione stringhe (`strcmp`, `strcpy`, `strlen`, `strcat`)
- Sfruttare funzioni matematiche avanzate (`sqrt`, `pow`, `sin`, `cos`, `tan`, `log`, `exp`)
- Gestire il tempo e le date (`time`, `difftime`, `localtime`, `gmtime`)
- Allocare e liberare memoria dinamica (`malloc`, `free`, `realloc`, `calloc`)
- Ordinare e cercare dati (`qsort`, `bsearch`)
- Implementare algoritmi efficienti usando le librerie standard

## 14.2 string.h - Manipolazione di Stringhe

### 14.2.1 Funzioni Base

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(int argc, char** argv) {
5     // strlen - lunghezza della stringa
6     char stringa[] = "Ciao";
7     printf("Lunghezza: %lu\n", strlen(stringa));
8
9     // strcmp - confronto tra stringhe
10    char str1[] = "abc";
11    char str2[] = "abc";
12    char str3[] = "def";
13
14    if (strcmp(str1, str2) == 0) {
15        printf("str1 e str2 sono uguali\n");
16    }
17
18    if (strcmp(str1, str3) < 0) {
19        printf("str1 viene prima di str3 alfabeticamente\n");
20    }
21
22    return 0;
23 }
```

### 14.2.2 Copia di Stringhe

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(int argc, char** argv) {
5     char sorgente[] = "Programmazione";
6     char destinazione[50];
7
8     // strcpy - copia una stringa
9     strcpy(destinazione, sorgente);
10    printf("Copia: %s\n", destinazione);
11
12    // strncpy - copia massimo n caratteri (piu' sicuro)
13    char buffer[10];
14    strncpy(buffer, sorgente, 9);
15    buffer[9] = '\0'; // Assicura terminazione
16    printf("Copia limitata: %s\n", buffer);
17
18    return 0;
19 }
```

### 14.2.3 Concatenazione e Ricerca

```
1 #include <stdio.h>
2 #include <string.h>
```

```

3
4 int main(int argc, char** argv) {
5     char testo[] = "Ciao Mondo";
6
7     // strchr - trova primo carattere
8     char* pos = strchr(testo, 'M');
9     if (pos != NULL) {
10         printf("'M' trovato a posizione %ld\n", pos - testo);
11     }
12
13     // strstr - trova sottostringa
14     char* sottostringaPos = strstr(testo, "Mondo");
15     if (sottostringaPos != NULL) {
16         printf("Sottostringa trovata: %s\n", sottostringaPos);
17     }
18
19     // strcat - concatena stringhe
20     char str1[50] = "Ciao ";
21     char str2[] = "Mondo";
22     strcat(str1, str2);
23     printf("Concatenazione: %s\n", str1);
24
25     // strncat - concatena massimo n caratteri
26     char str3[50] = "Buon ";
27     strncat(str3, "Pomeriggio Amico", 8);
28     printf("Concatenazione limitata: %s\n", str3);
29
30     return 0;
31 }

```

#### 14.2.4 Funzioni Avanzate

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 int main(int argc, char** argv) {
6     // strcspn - lunghezza prima di caratteri specifici
7     char testo[] = "Ciao, Mondo!";
8     int pos = strcspn(testo, ",!");
9     printf("Posizione punteggiatura: %d\n", pos);
10
11     // strpbrk - trova primo di vari caratteri
12     char* p = strpbrk(testo, ",!");
13     if (p != NULL) {
14         printf("Primo carattere speciale: %c\n", *p);
15     }
16
17     // strtok - divide una stringa (distruttivo!)
18     char copia[50] = "Uno,Due,Tre,Quattro";
19     char* token = strtok(copia, ",");
20
21     while (token != NULL) {
22         printf("Token: %s\n", token);
23         token = strtok(NULL, ",");
24     }
25 }

```

```
26     return 0;
27 }
```

### Attenzione

`strtok` modifica la stringa originale inserendo `'\0'`. Non usare su stringhe costanti o che devi riutilizzare.

## 14.3 math.h - Operazioni Matematiche

### 14.3.1 Funzioni Trigonometriche

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char** argv) {
5     // Conversione da gradi a radianti
6     double gradi = 45.0;
7     double radianti = gradi * M_PI / 180.0;
8
9     printf("Seno di 45$^\circ$: %.4f\n", sin(radianti));
10    printf("Coseno di 45$^\circ$: %.4f\n", cos(radianti));
11    printf("Tangente di 45$^\circ$: %.4f\n", tan(radianti));
12
13    // Funzioni inverse
14    printf("Arcsin(0.5): %.4f rad\n", asin(0.5));
15    printf("Arccos(0.5): %.4f rad\n", acos(0.5));
16    printf("Arctan(1.0): %.4f rad\n", atan(1.0));
17
18    // atan2(y, x) - calcola angolo nel piano cartesiano
19    printf("atan2(1, 1): %.4f rad\n", atan2(1.0, 1.0));
20
21    return 0;
22 }
```

Compila con: `gcc -lm -o programma main.c`

### 14.3.2 Potenze e Radici

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char** argv) {
5     // pow(base, esponente) - elevamento a potenza
6     printf("2^10 = %.0f\n", pow(2, 10));
7     printf("3^3 = %.0f\n", pow(3, 3));
8     printf("5^0.5 = %.4f\n", pow(5, 0.5));
9
10    // sqrt(x) - radice quadrata
11    printf("Radice di 16: %.2f\n", sqrt(16.0));
12    printf("Radice di 2: %.4f\n", sqrt(2.0));
13
14    // cbrt(x) - radice cubica
15    printf("Radice cubica di 27: %.2f\n", cbrt(27.0));
```

```
16 // hypot(x, y) - ipotenusa: sqrt(x^2 + y^2)
17 printf("Ipotenusa(3, 4): %.2f\n", hypot(3.0, 4.0));
18
19 return 0;
20 }
21
```

### 14.3.3 Logaritmi e Funzioni Esponenziali

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char** argv) {
5     // exp(x) - e^x
6     printf("e^1 = %.4f\n", exp(1.0));
7     printf("e^2 = %.4f\n", exp(2.0));
8
9     // log(x) - logaritmo naturale (base e)
10    printf("ln(10) = %.4f\n", log(10.0));
11
12    // log10(x) - logaritmo in base 10
13    printf("log10(100) = %.2f\n", log10(100.0));
14
15    // log2(x) - logaritmo in base 2
16    printf("log2(8) = %.2f\n", log2(8.0));
17
18    return 0;
19 }
```

### 14.3.4 Funzioni di Arrotondamento e Valore Assoluto

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char** argv) {
5     double num = 3.7;
6
7     // floor - arrotonda per difetto
8     printf("floor(3.7) = %.0f\n", floor(num));
9
10    // ceil - arrotonda per eccesso
11    printf("ceil(3.7) = %.0f\n", ceil(num));
12
13    // round - arrotonda al piu' vicino
14    printf("round(3.7) = %.0f\n", round(num));
15
16    // trunc - trunca la parte decimale
17    printf("trunc(3.7) = %.0f\n", trunc(num));
18
19    // fabs - valore assoluto (float/double)
20    printf("fabs(-5.5) = %.1f\n", fabs(-5.5));
21
22    // Resto della divisione con decimali
23    printf("fmod(7.5, 2.5) = %.1f\n", fmod(7.5, 2.5));
24 }
```

```
25     return 0;
26 }
```

## 14.4 time.h - Gestione del Tempo

### 14.4.1 Funzioni Base di Tempo

```
1  #include <stdio.h>
2  #include <time.h>
3
4  int main(int argc, char** argv) {
5      // time(NULL) - secondi dal 1 gennaio 1970
6      time_t ora_attuale = time(NULL);
7      printf("Secondi da epoch: %ld\n", (long)ora_attuale);
8
9      // ctime - converte a stringa leggibile
10     printf("Data e ora: %s", ctime(&ora_attuale));
11
12     // localtime - scompone in struttura
13     struct tm* info_tempo = localtime(&ora_attuale);
14
15     printf("Anno: %d\n", info_tempo->tm_year + 1900);
16     printf("Mese: %d\n", info_tempo->tm_mon + 1);
17     printf("Giorno: %d\n", info_tempo->tm_mday);
18     printf("Ora: %d\n", info_tempo->tm_hour);
19     printf("Minuti: %d\n", info_tempo->tm_min);
20     printf("Secondi: %d\n", info_tempo->tm_sec);
21
22     return 0;
23 }
```

### 14.4.2 Misurazione del Tempo Trascorso

```
1  #include <stdio.h>
2  #include <time.h>
3
4  int main(int argc, char** argv) {
5      // Metodo 1: clock()
6      clock_t inizio = clock();
7
8      // Codice da misurare
9      int somma = 0;
10     for (int i = 0; i < 1000000; i++) {
11         somma += i;
12     }
13
14     clock_t fine = clock();
15     double tempo_cpu = (double)(fine - inizio) / CLOCKS_PER_SEC;
16     printf("Tempo CPU: %.4f secondi\n", tempo_cpu);
17
18     // Metodo 2: time() per tempo reale
19     time_t t1 = time(NULL);
20
21     sleep(2); // Aspetta 2 secondi
```

```

22
23     time_t t2 = time(NULL);
24     printf("Tempo reale: %ld secondi\n", (long)(t2 - t1));
25
26     return 0;
27 }

```

### 14.4.3 Formattazione della Data

```

1  #include <stdio.h>
2  #include <time.h>
3
4  int main(int argc, char** argv) {
5      time_t ora = time(NULL);
6      struct tm* info = localtime(&ora);
7
8      char buffer[100];
9
10     // strftime - formatta la data/ora
11     strftime(buffer, sizeof(buffer), "%d/%m/%Y", info);
12     printf("Data: %s\n", buffer);
13
14     strftime(buffer, sizeof(buffer), "%H:%M:%S", info);
15     printf("Ora: %s\n", buffer);
16
17     strftime(buffer, sizeof(buffer), "%A, %d %B %Y", info);
18     printf("Data estesa: %s\n", buffer);
19
20     return 0;
21 }

```

#### Attenzione

I mesi in `tm` vanno da 0-11 (gennaio=0), gli anni vanno aggiunti a 1900. Questo è un comune source di bug!

## 14.5 stdlib.h - Allocazione Memoria e Utilità

### 14.5.1 Allocazione Dinamica

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv) {
5      // malloc - alloca memoria non inizializzata
6      int* array = (int*)malloc(10 * sizeof(int));
7
8      if (array == NULL) {
9          printf("Errore allocazione!\n");
10         return 1;
11     }
12
13     // Usa l'array

```

```
14     for (int i = 0; i < 10; i++) {
15         array[i] = i * 2;
16     }
17
18     for (int i = 0; i < 10; i++) {
19         printf("%d ", array[i]);
20     }
21     printf("\n");
22
23     // Libera memoria
24     free(array);
25     array = NULL;    // Buona pratica
26
27     return 0;
28 }
```

### 14.5.2 calloc e realloc

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv) {
5      // calloc - alloca e inizializza a zero
6      int* array1 = (int*)calloc(5, sizeof(int));
7
8      printf("Calloc (inizializzato a zero): ");
9      for (int i = 0; i < 5; i++) {
10         printf("%d ", array1[i]);
11     }
12     printf("\n");
13
14     // realloc - ridimensiona un blocco di memoria
15     int* array2 = (int*)malloc(3 * sizeof(int));
16     array2[0] = 10;
17     array2[1] = 20;
18     array2[2] = 30;
19
20     // Espandi a 6 elementi
21     array2 = (int*)realloc(array2, 6 * sizeof(int));
22
23     if (array2 == NULL) {
24         printf("Errore realloc!\n");
25         return 1;
26     }
27
28     printf("Dopo realloc: ");
29     for (int i = 0; i < 6; i++) {
30         printf("%d ", array2[i]);
31     }
32     printf("\n");
33
34     free(array1);
35     free(array2);
36
37     return 0;
38 }
```

### 14.5.3 Ordinamento e Ricerca

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Funzione di confronto per qsort (interi)
5 int compara_interi(const void* a, const void* b) {
6     int val_a = *(int*)a;
7     int val_b = *(int*)b;
8
9     if (val_a < val_b) return -1;
10    if (val_a > val_b) return 1;
11    return 0;
12 }
13
14 // Funzione di confronto per stringhe
15 int compara_stringhe(const void* a, const void* b) {
16     return strcmp(*(char**)a, *(char**)b);
17 }
18
19 int main(int argc, char** argv) {
20     // qsort - ordinamento generico
21     int array[] = {64, 34, 25, 12, 22, 11, 90};
22     int n = 7;
23
24     printf("Prima: ");
25     for (int i = 0; i < n; i++) {
26         printf("%d ", array[i]);
27     }
28     printf("\n");
29
30     // Ordina l'array
31     qsort(array, n, sizeof(int), compara_interi);
32
33     printf("Dopo: ");
34     for (int i = 0; i < n; i++) {
35         printf("%d ", array[i]);
36     }
37     printf("\n");
38
39     return 0;
40 }
```

### 14.5.4 bsearch - Ricerca Binaria

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int compara_interi(const void* a, const void* b) {
5     return *(int*)a - *(int*)b;
6 }
7
8 int main(int argc, char** argv) {
9     int array[] = {10, 20, 30, 40, 50, 60, 70};
10    int n = 7;
11 }
```

```

12 // L'array DEVE essere ordinato per bsearch
13 qsort(array, n, sizeof(int), compara_interi);
14
15 int valore_cercato = 40;
16 int* risultato = (int*)bsearch(&valore_cercato, array, n,
17                               sizeof(int), compara_interi);
18
19 if (risultato != NULL) {
20     printf("Trovato: %d\n", *risultato);
21 } else {
22     printf("Non trovato\n");
23 }
24
25 return 0;
26 }

```

## 14.6 Esempio Completo: Gestione Database di Studenti

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5  #include <time.h>
6
7  typedef struct {
8      int id;
9      char nome[30];
10     float voti[5];
11     int num_voti;
12     time_t data_iscrizione;
13 } Studente;
14
15 // Calcola media dei voti
16 float calcola_media(Studente* s) {
17     if (s->num_voti == 0) return 0.0;
18
19     float somma = 0.0;
20     for (int i = 0; i < s->num_voti; i++) {
21         somma += s->voti[i];
22     }
23
24     return somma / s->num_voti;
25 }
26
27 // Deviazione standard
28 float calcola_deviazione(Studente* s) {
29     if (s->num_voti <= 1) return 0.0;
30
31     float media = calcola_media(s);
32     float somma_quadrati = 0.0;
33
34     for (int i = 0; i < s->num_voti; i++) {
35         float diff = s->voti[i] - media;
36         somma_quadrati += diff * diff;
37     }
38

```

```

39     return sqrt(somma_quadrati / (s->num_voti - 1));
40 }
41
42 // Confronto per qsort
43 int compara_studenti(const void* a, const void* b) {
44     float media_a = calcola_media((Studente*)a);
45     float media_b = calcola_media((Studente*)b);
46
47     if (media_a > media_b) return -1;
48     if (media_a < media_b) return 1;
49     return 0;
50 }
51
52 int main(int argc, char** argv) {
53     // Alloca array di studenti
54     int num_studenti = 3;
55     Studente* studenti = (Studente*)malloc(num_studenti * sizeof(
        Studente));
56
57     // Popola dati
58     studenti[0] = {1, "Mario Rossi", {28, 27, 25}, 3, time(NULL)};
59     studenti[1] = {2, "Lucia Bianchi", {30, 29, 28}, 3, time(NULL)};
60     studenti[2] = {3, "Giovanni Verdi", {20, 22, 21}, 3, time(NULL)};
61
62     // Ordina per media (decrescente)
63     qsort(studenti, num_studenti, sizeof(Studente), compara_studenti);
64
65     printf("=== STATISTICHE STUDENTI ===\n");
66     for (int i = 0; i < num_studenti; i++) {
67         float media = calcola_media(&studenti[i]);
68         float dev = calcola_deviazione(&studenti[i]);
69
70         printf("\n%s:\n", studenti[i].nome);
71         printf("  Media: %.2f\n", media);
72         printf("  Deviazione: %.2f\n", dev);
73         printf("  Voti: ");
74         for (int j = 0; j < studenti[i].num_voti; j++) {
75             printf("%.0f ", studenti[i].voti[j]);
76         }
77         printf("\n");
78     }
79
80     free(studenti);
81     return 0;
82 }

```

### Attenzione

Quando allochi memoria con `malloc`, devi sempre:

- Controllare che il puntatore non sia `NULL`
- Liberare la memoria con `free` quando finisci
- Non accedere a memoria già liberata
- Non fare doppi `free`

## 14.7 Esercizi

### 14.7.1 Livello Base

1. Scrivi un programma che legge due stringhe, le confronta con `strcmp`, e stampa quale viene prima alfabeticamente.
2. Crea un programma che calcola seno, coseno, tangente di un angolo fornito in gradi.
3. Implementa un programma che stampa la data e l'ora attuali in formato leggibile usando `ctime`.
4. Scrivi un programma che alloca dinamicamente un array di 10 interi, lo riempie, e poi lo libera.

### 14.7.2 Livello Intermedio

1. Crea un programma che prende una stringa come input, la divide usando `strtok`, e stampa i token.
2. Implementa una calcolatrice scientifica che usa funzioni da `math.h` per seno, coseno, sqrt, pow, etc.
3. Scrivi un programma che misura il tempo di esecuzione di diversi algoritmi di ordinamento.
4. Crea un programma che ordina un array di numeri casuali usando `qsort` e poi cerca un valore con `bsearch`.

### 14.7.3 Livello Avanzato

1. Implementa un sistema di gestione studenti che calcola media, mediana, deviazione standard dei voti usando `stdlib.h` e `math.h`.
2. Crea un programma che legge un file CSV di dati, alloca memoria dinamica, ordina i dati, e li salva in un nuovo file.
3. Scrivi un programma che misura prestazioni di memoria: alloca, ridimensiona, e libera blocchi usando `malloc`, `realloc`, `free`.
4. Implementa una struttura dati dinamica (lista collegata) che usa `malloc/free` e supporta operazioni di ricerca/ordinamento.

## 14.8 Riepilogo

- `string.h`: `strlen`, `strcmp`, `strcpy`, `strcat`, `strchr`, `strstr`, `strtok`
- `math.h`: trigonometriche (sin, cos, tan), potenze (pow, sqrt), logaritmi (log, log10), arrotondamento (floor, ceil, round)
- Compila con `gcc -lm` per linkare la libreria matematica
- `time.h`: `time()`, `ctime()`, `localtime()`, `strftime()` per date/ore
- `stdlib.h`: `malloc`, `calloc`, `realloc`, `free` per memoria dinamica

- `qsort` per ordinamento generico con funzione di comparazione
- `bsearch` per ricerca binaria (array deve essere ordinato)
- Sempre controllare il ritorno di `malloc` (NULL = errore)
- Sempre liberare memoria allocata dinamicamente
- `strtok` modifica la stringa originale - usa su una copia
- I mesi da 0-11, gli anni aggiunti a 1900 in `struct tm`

# Bibliografia e Risorse

## Libri di Testo

### Testi Fondamentali

1. **Kernighan, B. W., & Ritchie, D. M.** (1988). *The C Programming Language* (2nd ed.). Prentice Hall.
  - Il libro classico scritto dai creatori del linguaggio C
  - Considerato la "bibbia" del C
  - Copre tutte le caratteristiche fondamentali del linguaggio
2. **King, K. N.** (2008). *C Programming: A Modern Approach* (2nd ed.). W. W. Norton & Company.
  - Approccio moderno alla programmazione in C
  - Molti esempi pratici ed esercizi
  - Ottimo per principianti
3. **Deitel, P., & Deitel, H.** (2015). *C How to Program* (8th ed.). Pearson.
  - Trattazione completa con approccio didattico
  - Numerosi esempi e case study
  - Include esercizi gradualmente

### Testi Avanzati

1. **Kochan, S. G.** (2014). *Programming in C* (4th ed.). Addison-Wesley.
2. **Prata, S.** (2013). *C Primer Plus* (6th ed.). Addison-Wesley.
3. **Summit, S.** (1995). *C Programming FAQs*. Addison-Wesley.

## Risorse Online

### Tutorial e Guide

1. **Learn-C.org**
  - <https://www.learn-c.org/>
  - Tutorial interattivo gratuito

- Esercizi pratici con verifica automatica

## 2. GeeksforGeeks - C Programming

- <https://www.geeksforgeeks.org/c-programming-language/>
- Articoli approfonditi su ogni aspetto del C
- Molti esempi e spiegazioni

## 3. Tutorialspoint - C Programming

- <https://www.tutorialspoint.com/cprogramming/>
- Tutorial completo con esempi
- Possibilità di compilare online

## 4. C Programming at Programiz

- <https://www.programiz.com/c-programming>
- Guide chiare per principianti
- Esempi pratici e ben commentati

## Documentazione di Riferimento

### 1. cppreference.com

- <https://en.cppreference.com/w/c>
- Documentazione completa e accurata
- Include C11 e C17

### 2. C Standard Library Reference

- <https://www.cplusplus.com/reference/clibrary/>
- Riferimento per tutte le funzioni standard
- Esempi per ogni funzione

## Piattaforme di Pratica

### 1. HackerRank

- <https://www.hackerrank.com/domains/c>
- Sfide di programmazione in C
- Livelli di difficoltà progressivi

### 2. LeetCode

- <https://leetcode.com/>
- Problemi di algoritmi e strutture dati
- Supporto per C

### 3. Codewars

- <https://www.codewars.com/>
- Kata (esercizi) di vari livelli

- Sistema di ranking e progressione

#### 4. Project Euler

- <https://projecteuler.net/>
- Problemi matematici da risolvere con programmazione
- Ottimo per migliorare il problem solving

#### 5. Codeforces

- <https://codeforces.com/>
- Competizioni di programmazione
- Archivio di problemi storici

## Strumenti di Sviluppo

### Compilatori

#### 1. GCC (GNU Compiler Collection)

- Disponibile su Linux, macOS, Windows
- Open source e molto diffuso
- <https://gcc.gnu.org/>

#### 2. Clang

- Compilatore moderno con ottimi messaggi di errore
- Integrato in Xcode su macOS
- <https://clang.llvm.org/>

#### 3. Microsoft Visual C++

- Compilatore Microsoft per Windows
- Integrato in Visual Studio

### IDE (Integrated Development Environment)

#### 1. Visual Studio Code

- Editor leggero e versatile
- Estensioni per C/C++
- <https://code.visualstudio.com/>

#### 2. Code::Blocks

- IDE gratuito specifico per C/C++
- Interfaccia semplice
- <http://www.codeblocks.org/>

#### 3. CLion

- IDE professionale di JetBrains
- Gratuito per studenti
- <https://www.jetbrains.com/clion/>

#### 4. Dev-C++

- IDE semplice per Windows
- Buono per principianti
- <https://www.bloodshed.net/devcpp.html>

## Compilatori Online

### 1. OnlineGDB

- <https://www.onlinegdb.com/>
- Compilatore e debugger online

### 2. Repl.it

- <https://replit.com/>
- Ambiente di sviluppo online completo

### 3. JDoodle

- <https://www.jdoodle.com/c-online-compiler>
- Compilatore C online semplice

## Debugger

### 1. GDB (GNU Debugger)

- Debugger a riga di comando
- Molto potente
- <https://www.gnu.org/software/gdb/>

### 2. Valgrind

- Analisi di memoria
- Rileva memory leak
- <https://valgrind.org/>

## Standard e Specifiche

### Standard C

#### 1. C89/C90 - ANSI C / ISO C

- Prima standardizzazione del C

#### 2. C99 - ISO/IEC 9899:1999

- Array a lunghezza variabile
  - Commenti in stile C++
  - Tipo `long long`
3. **C11** - ISO/IEC 9899:2011
    - Supporto multi-threading
    - Analisi statica migliorata
  4. **C17/C18** - ISO/IEC 9899:2018
    - Correzioni e chiarimenti su C11

## Community e Forum

1. **Stack Overflow**
  - <https://stackoverflow.com/questions/tagged/c>
  - Domande e risposte sulla programmazione in C
2. **Reddit - r/C\_Programming**
  - [https://www.reddit.com/r/C\\_Programming/](https://www.reddit.com/r/C_Programming/)
  - Community attiva di programmatori C
3. **C Board**
  - <https://cboard.cprogramming.com/>
  - Forum dedicato alla programmazione in C

## Canali YouTube

1. **freeCodeCamp.org** - Tutorial completi C
2. **The Cherno** - Programmazione C/C++
3. **Jacob Sorber** - C programming concepts
4. **Code Vault** - Advanced C Programming

## Best Practices e Style Guide

1. **GNU Coding Standards**
  - <https://www.gnu.org/prep/standards/>
  - Standard di codifica GNU
2. **Linux Kernel Coding Style**
  - <https://www.kernel.org/doc/html/latest/process/coding-style.html>
  - Style guide del kernel Linux

### 3. MISRA C

- Standard per C sicuro in sistemi critici
- Usato nell'automotive e aerospace

## Progetti Open Source in C

Studiare codice di qualità è un ottimo modo per imparare:

1. **Linux Kernel** - <https://github.com/torvalds/linux>
2. **Git** - <https://github.com/git/git>
3. **SQLite** - <https://www.sqlite.org/>
4. **Redis** - <https://github.com/redis/redis>
5. **Nginx** - <https://github.com/nginx/nginx>

## Certificazioni

1. **C Programming Language Certified Associate (CLA)**
  - Certificazione entry-level
  - Offerta da C++ Institute
2. **C Programming Language Certified Professional (CLP)**
  - Certificazione avanzata
  - Per programmatori esperti

## Note Finali

Questa bibliografia rappresenta solo un punto di partenza. Il mondo della programmazione in C è vasto e in continua evoluzione. Alcuni consigli:

- **Pratica costante:** la programmazione si impara programmando
- **Leggi codice:** studia progetti open source ben scritti
- **Partecipa alle community:** fai domande e aiuta altri
- **Mantieniti aggiornato:** segui blog e newsletter
- **Sperimenta:** non aver paura di provare cose nuove

Ricorda: il miglior programmatore non è quello che conosce tutto, ma quello che sa dove trovare le informazioni quando serve!

# Appendice: Soluzioni ad Alcuni Esercizi

Questa appendice contiene le soluzioni di alcuni esercizi selezionati dai capitoli precedenti. Per ogni esercizio viene fornita una soluzione commentata.

## Capitolo 2: Variabili e Tipi di Dati

### Esercizio Base 2: Area di un Cerchio

```
1 #include <stdio.h>
2 #define PI 3.14159
3
4 int main() {
5     float raggio, area;
6
7     printf("Inserisci il raggio del cerchio: ");
8     scanf("%f", &raggio);
9
10    // Formula: area = PI * r^2
11    area = PI * raggio * raggio;
12
13    printf("L'area del cerchio e': %.2f\n", area);
14
15    return 0;
16 }
```

### Esercizio Intermedio 1: Secondi in Ore, Minuti, Secondi

```
1 #include <stdio.h>
2
3 int main() {
4     int secondi_totali, ore, minuti, secondi;
5
6     printf("Inserisci il numero di secondi: ");
7     scanf("%d", &secondi_totali);
8
9     ore = secondi_totali / 3600;
10    minuti = (secondi_totali % 3600) / 60;
11    secondi = secondi_totali % 60;
12
13    printf("%d secondi corrispondono a:\n", secondi_totali);
14    printf("%d ore, %d minuti, %d secondi\n", ore, minuti, secondi);
15 }
```

```
15
16     return 0;
17 }
```

## Capitolo 3: Operatori ed Espressioni

### Esercizio Base 3: Pari o Dispari

```
1 #include <stdio.h>
2
3 int main() {
4     int numero;
5
6     printf("Inserisci un numero: ");
7     scanf("%d", &numero);
8
9     if (numero % 2 == 0) {
10         printf("%d e' pari\n", numero);
11     } else {
12         printf("%d e' dispari\n", numero);
13     }
14
15     return 0;
16 }
```

### Esercizio Intermedio 3: Anno Bisestile

```
1 #include <stdio.h>
2
3 int main() {
4     int anno;
5
6     printf("Inserisci un anno: ");
7     scanf("%d", &anno);
8
9     // Un anno e' bisestile se:
10    // - divisibile per 4 E
11    // - (non divisibile per 100 OPPURE divisibile per 400)
12    if ((anno % 4 == 0) && (anno % 100 != 0 || anno % 400 == 0)) {
13        printf("%d e' un anno bisestile\n", anno);
14    } else {
15        printf("%d non e' un anno bisestile\n", anno);
16    }
17
18    return 0;
19 }
```

## Capitolo 4: Controllo di Flusso

### Esercizio Intermedio 1: Numero Primo

```
1 #include <stdio.h>
2
3 int main() {
4     int numero, i, primo = 1;
5
6     printf("Inserisci un numero: ");
7     scanf("%d", &numero);
8
9     if (numero <= 1) {
10         primo = 0;
11     } else {
12         for (i = 2; i * i <= numero; i++) {
13             if (numero % i == 0) {
14                 primo = 0;
15                 break;
16             }
17         }
18     }
19
20     if (primo) {
21         printf("%d e' un numero primo\n", numero);
22     } else {
23         printf("%d non e' un numero primo\n", numero);
24     }
25
26     return 0;
27 }
```

### Esercizio Intermedio 2: Inversione di un Numero

```
1 #include <stdio.h>
2
3 int main() {
4     int numero, inverso = 0, cifra;
5
6     printf("Inserisci un numero: ");
7     scanf("%d", &numero);
8
9     int originale = numero;
10
11     while (numero != 0) {
12         cifra = numero % 10;
13         inverso = inverso * 10 + cifra;
14         numero /= 10;
15     }
16
17     printf("Il numero %d invertito e': %d\n", originale, inverso);
18
19     return 0;
20 }
```

### Esercizio Avanzato 1: Sequenza di Fibonacci

```
1 #include <stdio.h>
```

```
2
3 int main() {
4     int n, i;
5     int primo = 0, secondo = 1, successivo;
6
7     printf("Quanti termini della sequenza Fibonacci? ");
8     scanf("%d", &n);
9
10    printf("Sequenza di Fibonacci:\n");
11
12    for (i = 0; i < n; i++) {
13        if (i <= 1) {
14            successivo = i;
15        } else {
16            successivo = primo + secondo;
17            primo = secondo;
18            secondo = successivo;
19        }
20        printf("%d ", successivo);
21    }
22    printf("\n");
23
24    return 0;
25 }
```

## Capitolo 5: Funzioni

### Esercizio Base 2: Verifica Numero Pari

```
1 #include <stdio.h>
2
3 int is_pari(int n) {
4     return (n % 2 == 0);
5 }
6
7 int main() {
8     int numero;
9
10    printf("Inserisci un numero: ");
11    scanf("%d", &numero);
12
13    if (is_pari(numero)) {
14        printf("%d e' pari\n", numero);
15    } else {
16        printf("%d e' dispari\n", numero);
17    }
18
19    return 0;
20 }
```

### Esercizio Intermedio 1: Inversione Numero con Funzione

```
1 #include <stdio.h>
2
```

```
3 int inverti_numero(int n) {
4     int inverso = 0;
5
6     while (n != 0) {
7         inverso = inverso * 10 + (n % 10);
8         n /= 10;
9     }
10
11     return inverso;
12 }
13
14 int main() {
15     int numero;
16
17     printf("Inserisci un numero: ");
18     scanf("%d", &numero);
19
20     int invertito = inverti_numero(numero);
21
22     printf("%d invertito e': %d\n", numero, invertito);
23
24     return 0;
25 }
```

## Esercizio Intermedio 2: Somma Cifre Ricorsiva

```
1 #include <stdio.h>
2
3 int somma_cifre(int n) {
4     // Caso base
5     if (n == 0) {
6         return 0;
7     }
8     // Caso ricorsivo
9     return (n % 10) + somma_cifre(n / 10);
10 }
11
12 int main() {
13     int numero;
14
15     printf("Inserisci un numero: ");
16     scanf("%d", &numero);
17
18     int somma = somma_cifre(numero);
19
20     printf("La somma delle cifre di %d e': %d\n", numero, somma);
21
22     return 0;
23 }
```

## Capitolo 6: Array

### Esercizio Base 1: Conta Pari e Dispari

```
1 #include <stdio.h>
2
3 int main() {
4     int numeri[10];
5     int pari = 0, dispari = 0;
6
7     printf("Inserisci 10 numeri:\n");
8     for (int i = 0; i < 10; i++) {
9         printf("Numero %d: ", i + 1);
10        scanf("%d", &numeri[i]);
11
12        if (numeri[i] % 2 == 0) {
13            pari++;
14        } else {
15            dispari++;
16        }
17    }
18
19    printf("\nNumeri pari: %d\n", pari);
20    printf("Numeri dispari: %d\n", dispari);
21
22    return 0;
23 }
```

### Esercizio Base 4: Array Palindromo

```
1 #include <stdio.h>
2
3 int main() {
4     int arr[] = {1, 2, 3, 2, 1};
5     int n = sizeof(arr) / sizeof(arr[0]);
6     int palindromo = 1;
7
8     for (int i = 0; i < n / 2; i++) {
9         if (arr[i] != arr[n - 1 - i]) {
10            palindromo = 0;
11            break;
12        }
13    }
14
15    if (palindromo) {
16        printf("L'array e' palindromo\n");
17    } else {
18        printf("L'array non e' palindromo\n");
19    }
20
21    return 0;
22 }
```

### Esercizio Intermedio 1: Ricerca Binaria

```
1 #include <stdio.h>
2
3 int ricerca_binaria(int arr[], int n, int valore) {
```

```
4     int sinistra = 0, destra = n - 1;
5
6     while (sinistra <= destra) {
7         int medio = sinistra + (destra - sinistra) / 2;
8
9         if (arr[medio] == valore) {
10             return medio;
11         }
12
13         if (arr[medio] < valore) {
14             sinistra = medio + 1;
15         } else {
16             destra = medio - 1;
17         }
18     }
19
20     return -1; // Non trovato
21 }
22
23 int main() {
24     int arr[] = {2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78};
25     int n = sizeof(arr) / sizeof(arr[0]);
26     int valore = 23;
27
28     int posizione = ricerca_binaria(arr, n, valore);
29
30     if (posizione != -1) {
31         printf("%d trovato alla posizione %d\n", valore, posizione);
32     } else {
33         printf("%d non trovato\n", valore);
34     }
35
36     return 0;
37 }
```

## Capitolo 7: Puntatori

### Esercizio Base 1: Scambio Ciclico

```
1 #include <stdio.h>
2
3 void scambio_ciclico(int *a, int *b, int *c) {
4     int temp = *a;
5     *a = *b;
6     *b = *c;
7     *c = temp;
8 }
9
10 int main() {
11     int x = 1, y = 2, z = 3;
12
13     printf("Prima: x=%d, y=%d, z=%d\n", x, y, z);
14     scambio_ciclico(&x, &y, &z);
15     printf("Dopo: x=%d, y=%d, z=%d\n", x, y, z);
16
17     return 0;
18 }
```

```
18 }
```

### Esercizio Base 3: Max e Min con Puntatori

```
1 #include <stdio.h>
2
3 void trova_max_min(int arr[], int n, int *max, int *min) {
4     *max = arr[0];
5     *min = arr[0];
6
7     for (int i = 1; i < n; i++) {
8         if (arr[i] > *max) {
9             *max = arr[i];
10        }
11        if (arr[i] < *min) {
12            *min = arr[i];
13        }
14    }
15 }
16
17 int main() {
18     int numeri[] = {45, 12, 78, 23, 67, 89, 34};
19     int n = sizeof(numeri) / sizeof(numeri[0]);
20     int max, min;
21
22     trova_max_min(numeri, n, &max, &min);
23
24     printf("Massimo: %d\n", max);
25     printf("Minimo: %d\n", min);
26
27     return 0;
28 }
```

## Capitolo 8: Stringhe

### Esercizio Base 1: Conta Vocali

```
1 #include <stdio.h>
2 #include <ctype.h>
3
4 int conta_vocali(char *str) {
5     int conta = 0;
6
7     for (int i = 0; str[i] != '\0'; i++) {
8         char c = tolower(str[i]);
9         if (c == 'a' || c == 'e' || c == 'i' ||
10             c == 'o' || c == 'u') {
11             conta++;
12         }
13     }
14
15     return conta;
16 }
17
```

```
18 int main() {
19     char frase[100];
20
21     printf("Inserisci una frase: ");
22     fgets(frase, sizeof(frase), stdin);
23
24     int vocali = conta_vocali(frase);
25
26     printf("Numero di vocali: %d\n", vocali);
27
28     return 0;
29 }
```

## Esercizio Intermedio 2: Anagrammi

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  int sono_anagrammi(char *str1, char *str2) {
6      int freq1[26] = {0}, freq2[26] = {0};
7
8      // Conta frequenza caratteri prima stringa
9      for (int i = 0; str1[i] != '\0'; i++) {
10         if (isalpha(str1[i])) {
11             freq1[tolower(str1[i]) - 'a']++;
12         }
13     }
14
15     // Conta frequenza caratteri seconda stringa
16     for (int i = 0; str2[i] != '\0'; i++) {
17         if (isalpha(str2[i])) {
18             freq2[tolower(str2[i]) - 'a']++;
19         }
20     }
21
22     // Confronta frequenze
23     for (int i = 0; i < 26; i++) {
24         if (freq1[i] != freq2[i]) {
25             return 0;
26         }
27     }
28
29     return 1;
30 }
31
32 int main() {
33     char str1[] = "listen";
34     char str2[] = "silent";
35
36     if (sono_anagrammi(str1, str2)) {
37         printf("\n%s\" e \"%s\" sono anagrammi\n", str1, str2);
38     } else {
39         printf("\n%s\" e \"%s\" non sono anagrammi\n", str1, str2);
40     }
41
42     return 0;
43 }
```

```
43 }
```

## Capitolo 9: Struct

### Esercizio Base 1: Struct Rettangolo

```
1 #include <stdio.h>
2
3 typedef struct {
4     float base;
5     float altezza;
6 } Rettangolo;
7
8 float area(Rettangolo r) {
9     return r.base * r.altezza;
10 }
11
12 float perimetro(Rettangolo r) {
13     return 2 * (r.base + r.altezza);
14 }
15
16 int main() {
17     Rettangolo r;
18
19     printf("Inserisci base: ");
20     scanf("%f", &r.base);
21     printf("Inserisci altezza: ");
22     scanf("%f", &r.altezza);
23
24     printf("\nArea: %.2f\n", area(r));
25     printf("Perimetro: %.2f\n", perimetro(r));
26
27     return 0;
28 }
```

### Esercizio Base 3: Conto Corrente

```
1 #include <stdio.h>
2
3 typedef struct {
4     int numero;
5     char intestatario[50];
6     float saldo;
7 } ContoCorrente;
8
9 void deposita(ContoCorrente *conto, float importo) {
10     if (importo > 0) {
11         conto->saldo += importo;
12         printf("Depositati %.2f euro\n", importo);
13         printf("Nuovo saldo: %.2f euro\n", conto->saldo);
14     } else {
15         printf("Importo non valido!\n");
16     }
17 }
```

```

18
19 void preleva(ContoCorrente *conto, float importo) {
20     if (importo > 0 && importo <= conto->saldo) {
21         conto->saldo -= importo;
22         printf("Prelevati %.2f euro\n", importo);
23         printf("Nuovo saldo: %.2f euro\n", conto->saldo);
24     } else {
25         printf("Operazione non consentita!\n");
26     }
27 }
28
29 void stampa_saldo(ContoCorrente conto) {
30     printf("\n=== Conto %d ===\n", conto.numero);
31     printf("Intestatario: %s\n", conto.intestatario);
32     printf("Saldo: %.2f euro\n", conto.saldo);
33 }
34
35 int main() {
36     ContoCorrente conto = {12345, "Mario Rossi", 1000.00};
37
38     stampa_saldo(conto);
39
40     deposita(&conto, 500.00);
41     preleva(&conto, 200.00);
42     preleva(&conto, 2000.00); // Saldo insufficiente
43
44     stampa_saldo(conto);
45
46     return 0;
47 }

```

## Capitolo 10: File

### Esercizio Base 3: Conta Carattere in File

```

1  #include <stdio.h>
2
3  int main() {
4      FILE *file;
5      char nome_file[100];
6      char carattere_cerca, c;
7      int conta = 0;
8
9      printf("Nome del file: ");
10     scanf("%s", nome_file);
11
12     printf("Carattere da cercare: ");
13     scanf(" %c", &carattere_cerca);
14
15     file = fopen(nome_file, "r");
16     if (file == NULL) {
17         printf("Impossibile aprire il file!\n");
18         return 1;
19     }
20
21     while ((c = fgetc(file)) != EOF) {

```

```
22     if (c == carattere_cerca) {
23         conta++;
24     }
25 }
26
27 fclose(file);
28
29 printf("Il carattere '%c' appare %d volte\n",
30        carattere_cerca, conta);
31
32 return 0;
33 }
```

### Esercizio Intermedio 3: Cifrario di Cesare

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  void cifra_file(char *input, char *output, int chiave) {
5      FILE *fin = fopen(input, "r");
6      FILE *fout = fopen(output, "w");
7
8      if (fin == NULL || fout == NULL) {
9          printf("Errore apertura file!\n");
10         return;
11     }
12
13     char c;
14     while ((c = fgetc(fin)) != EOF) {
15         if (isalpha(c)) {
16             char base = isupper(c) ? 'A' : 'a';
17             c = base + (c - base + chiave) % 26;
18         }
19         fputc(c, fout);
20     }
21
22     fclose(fin);
23     fclose(fout);
24     printf("File cifrato con successo!\n");
25 }
26
27 void decifra_file(char *input, char *output, int chiave) {
28     cifra_file(input, output, 26 - chiave);
29 }
30
31 int main() {
32     int scelta, chiave;
33     char input[100], output[100];
34
35     printf("1. Cifra file\n");
36     printf("2. Decifra file\n");
37     printf("Scelta: ");
38     scanf("%d", &scelta);
39
40     printf("File input: ");
41     scanf("%s", input);
42     printf("File output: ");
```

```
43     scanf("%s", output);
44     printf("Chiave (0-25): ");
45     scanf("%d", &chiave);
46
47     if (scelta == 1) {
48         cifra_file(input, output, chiave);
49     } else if (scelta == 2) {
50         decifra_file(input, output, chiave);
51     }
52
53     return 0;
54 }
```

## Note Finali

Queste soluzioni rappresentano solo uno dei possibili approcci per risolvere gli esercizi. Ci sono spesso modi alternativi, ugualmente validi, per affrontare un problema.

Quando studi queste soluzioni:

- Cerca di capire la logica, non solo copiare il codice
- Prova a modificare le soluzioni per renderle migliori
- Confronta le tue soluzioni con queste
- Impara dai diversi approcci utilizzati