Programmazione in linguaggio C

Antonio Lioy
(lioy @ polito.it)

Politecnico di Torino
Dip. di Automatica e Informatica

Informazioni generali sul corso

- **sito web del corso:**
  - **http://security.polito.it/~lioy/12bhd/**
  - **nota: si può sostituire la tilde con %7e ossia http://security.polito.it/%7elioy/12bhd/**
  - **slide del corso (PDF, richiede Acrobat)**
  - **strumenti di sviluppo (Quincy-2005, BorlandC 3.1)**
  - **vecchi temi d'esame**
  - **risultati degli esami**
  - **…**
- **esame:**
  - **scritto (2 ore) = programma in C più tre domande di teoria**

Testi

- **copia dei lucidi usati nel corso (sul sito web del corso)**
- **copia dei lucidi usati nelle videolezioni (sul portale)**
- **videolezioni del Prof. Mezzalama (sul portale)**
- **[teoria] P.J.Deitel, M.H.Deitel**
  "C - Corso completo di programmazione"
  Apogeo, 2010
  (teoria)
- **[teoria] B.W.Kernighan, D.M.Ritchie**
  "Il linguaggio C. Principi di programmazione e manuale di riferimento"
  Prentice Hall, 2004
- **[esercizi] S.Nocco, S.Quer**
  "Guida alla programmazione in linguaggio C"
  Clut, 2009

LAIB

- **orario:**
  - **dal 28/3/11 in via Boggio (LAIB 2B)**
  - **3 squadre 14:30-16, 16-17:30, 17:30-19**
  - **rispettare tassativamente la suddivisione in squadre, non sono permessi cambi**
- **attivazione di Quincy-2005:**
  - **"Menù Avvio" > "Programmi Vari" > "Compilatori"**
- **2 assistenti (borsisti) + Sanchez**
- **attività:**
  - **prova dei programmi sviluppati in aula**
  - **esercizi proposti sul sito dell'esame**

Strumenti di sviluppo

- **strumento semplice**
  - **Quincy 2005, sul sito web del corso (16.0 MB)**
  - **freeware, pienamente integrato in Windows**
- **strumento più complesso (e completo)**
  - **Code::Blocks (www.codeblocks.org, scaricare la versione completa con compilatore)**
  - **disponibile per Windows, Mac e Linux**
- **per i "veri programmatori":**
  - **compilatore / linker gcc e debugger gdb**
  - **da linea di comando (per Linux, Windows e Mac)**
  - **gcc/gdb è installato automaticamente sia con Quincy-2005 sia con Code::Blocks**

Caratteristiche dei linguaggi di programmazione

- **ogni linguaggio di programmazione ad alto livello (HLL) è caratterizzato da:**
  - **tipi di dato (primitivi), es. int**
  - **tipi di dato (complessi), es. vettori**
  - **operazioni base, es. +**
  - **controllo del flusso delle operazioni, es. if**
- **ogni HLL ha una precisa grammatica e sintassi da rispettare:**
  - **parole chiave o "keyword"**
    - **riservate ad uno scopo specifico**
  - **identificatori**
    - **nomi logici per dati o operazioni**

Scrittura di un programma C

- **il file sorgente di un programma in linguaggio C deve essere costituito da caratteri US-ASCII (non sono consentiti i caratteri ASCII estesi, quali le lettere accentate)**
- **il linguaggio C è "case-sensitive" ossia distingue tra maiuscole e minuscole**
  - **ad esempio, le parole "Totale" e "totale" identificano entità diverse in C**

Struttura di un programma C

- **un programma C deve contenere almeno una funzione chiamata main( )**
  - **è la parte principale del programma**
  - **attivata dal loader quando si esegue il programma**
- **la funzione deve essere di tipo:**
  - **int**
    restituisce al S.O. un valore intero quale flag di stato sulla terminazione del programma
- **le istruzioni che costituiscono il main vanno elencate sequenzialmente tra graffe { }**

vuoto_i.c

Return ?

- **per restituire un valore al S.O. si può usare l'istruzione**
  return *codice_di_uscita* ;
- **il codice di uscita deve essere un numero intero**
- **per convenzione un codice di uscita pari a zero indica che il programma è terminato normalmente, mentre si usa un codice diverso da zero nel caso di terminazione dovuta ad un errore**
- **il valore può essere visualizzato a livello di S.O. con comandi che dipendono dal S.O.**
- **esempi:**
  - **(MS-DOS/Windows)**
    variabile %errorlevel% nei file batch
  - **(Unix/Linux/Mac)**
    variabile $? negli script bash

vuoto.bat

Uso di Quincy-2005

- **creazione di un file sorgente (e salvataggio!):**
  - **File > New > (C source file)**
  - **File > Save as > (scegliere cartella e dare nome – xxx.c)**
- **uso di un programma già presente in un file;**
  - **File > Open > (scegliere cartella e file)**
- **far eseguire il programma direttamente in un passo solo:**
  - **Project > Execute**
- **far eseguire il programma passo passo:**
  - **Debug > Step / Step Over / Step To Cursor**
- **creare un file eseguibile:**
  - **Project > Build**
  - **eseguo il programma lanciando il suo file .EXE da una finestra di comando**

Uso di gcc

- **usando un editor di testo (es. notepad, blocco note) creo il file sorgente:**
  - **`prova.c`**
- **traduco il file sorgente in file oggetto:**
  - **`gcc -c prova.c`**
  - **viene generato il file oggetto prova.o**
- **creo il file eseguibile tramite il linker:**
  - **`gcc -o prova.exe prova.o`**
  - **viene generato il file oggetto prova.exe**

- **oppure faccio tutto in un passo solo:**
  - **`gcc –o prova.exe prova.c`**

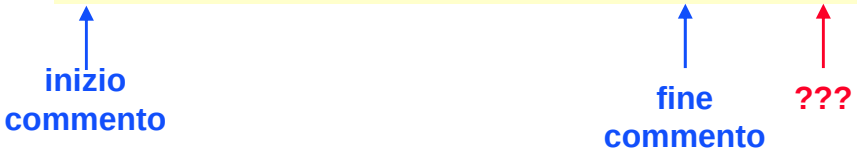
Commenti

- **un commento è un'informazione per chi legge il programma, non per il computer**
- **normalmente inserito a scopo di documentazione**
- **commenti (originali):**
  - **sono racchiusi tra `/*` e `*/`**
  - **possono comprendere varie righe**
  - **non possono essere "annidati" (un commento non può contenere al suo interno un altro commento)**
- **commenti (nuova forma, C99):**
  - **iniziano con `//`**
  - **terminano a fine riga**

Esempi di commenti

```
/*
un primo commento valido
*/

// un secondo commento valido (solo in C99)
```

```
/* commento che causa /* errore */ ... */
```



nullprog.c

Identificatori

- **un identificatore serve per far riferimento simbolicamente ad un oggetto, ossia tramite un nome invece che tramite l'indirizzo della cella di memoria in cui è conservato**
- **identificatori sono usati per identificare:**
  - **costanti (dati che non possono cambiare)**
  - **variabili (dati che possono cambiare)**
  - **tipi di dati (codifica dei dati)**
  - **funzioni (insiemi di istruzioni)**
  - **file (una sequenza di dati esterni al programma)**
  - **etichette o *label* (punti del programma)**

Caratteristiche di un identificatore

- **composto da uno o più caratteri**
- **inizia con carattere alfabetico o "_"**
- **contiene caratteri alfabetici, numerici o "_"**

- **esempi:**

```
PagaOraria

paga_oraria

_24

PIGRECO
```

Caratteristiche variabili di un identificatore

- **identificatori interni:**
  - **distinzione tra caratteri minuscoli e maiuscoli**
  - **significativi almeno i primi 31 caratteri**
- **identificatori esterni:**
  - **nessuna distinzione tra caratteri minuscoli e maiuscoli**
  - **significativi almeno i primi 6 caratteri**
- **identificatori riservati:**
  - **tutte le *keyword* del linguaggio (es. int, float, while, if)**
  - **gli elementi globali della libreria C standard (es. putchar, sin, clock)**

Keyword

- **C standard (K&R):**

```
auto      double    int       struct
break     else      long      switch
case      enum      register  typedef
char      extern    return    union
const     float     short     unsigned
continue  for       signed    void
default   goto      sizeof    volatile
do        if        static    while
```

- **il C99 ne aggiunge altre cinque:**

```
inline    restrict  _Bool     _Complex  _Imaginary
```

Definizione di variabili

- **ogni variabile rappresenta simbolicamente delle celle di memoria a lettura / scrittura:**
  - **il tipo di una variabile indica la codifica binaria da usarsi per memorizzarne il valore**
  - **il nome di una variabile indica l'indirizzo della cella di memoria ove è memorizzato il dato corrispodente**
- **si possono dichiarare più variabili dello stesso tipo separandole con una virgola**
- **sintassi:**

  ***tipo nome_variabile* [= *valore_iniziale*] ;**

- **esempi:**

```
int totale = 0;
float PrezzoLordo, PrezzoNetto;
```

Tipi base

- **i tipi base del linguaggio C sono:**
  - **char (caratteri ASCII)**
  - **int (numeri interi con segno)**
  - **float (numeri floating-point in singola precisione)**
  - **double (numeri floating-point in doppia precisione)**
  - **_Bool (valori Booleani) solo in C99**

Modificatori

- **i tipi possono essere preceduti da un modificatore:**
  - **signed (numeri con segno)**
  - **unsigned (numeri senza segno)**
  - **short (occupazione ridotta di memoria)**
  - **long (occupazione maggiore di memoria)**
- **nel C99 anche:**
  - **long long (occupazione molto maggiore di memoria)**
  - **_Complex (numeri complessi)**
  - **_Imaginary (numeri immaginari)**
- **il tipo int può essere sottinteso quando preceduto da un modificatore (es. "unsigned" invece di "unsigned int")**

Applicazione dei modificatori

- **[signed | unsigned] char**
- **[signed | unsigned] short [int]**
- **[signed | unsigned] int**
- **[signed | unsigned] long [int]**
- **[signed | unsigned] long long [int]**
- **float**
- **[long] double**
- **float _Complex**
- **[long] double _Complex**
- **float _Imaginary**
- **[long] double _ Imaginary**

Costanti (letterali)

tipo	*esempi*	
numeri interi	`26`	`-26`
numeri interi (long)	`26L`	`-26L`
numeri interi (unsigned)	`26U`	
numeri interi (unsigned long)	`26UL`	
numeri ottali	`032`	
numeri esadecimali	`0x1a`	`0X1A`
numeri floating-point	`230.4` `2.304e2`	`-230.4` `2.304E2`
singolo carattere	`'a'`	
stringhe di caratteri	`"alfabeto"`	

Costanti speciali di tipo carattere

carattere	*formato*
BEL (beep)	`\a`
BS (backspace)	`\b`
FF (form feed)	`\f`
LF (line feed)	`\n`
CR (carriage return)	`\r`
HT (horizontal tab)	`\t`
carattere ASCII (ottale)	`\nnn`
carattere ASCII (esadecimale)	`\Xnn`

Ritorno a capo

- la codifica dei file di testo usa schemi diversi per indicare che una riga è terminata e ne inizia una nuova:
  - MS-DOS / MS-Windows = CR+LF
  - UNIX = LF
  - MACINTOSH = CR
- per convenzione in linguaggio C si usa sempre e solo LF, ossia il carattere `\n`, lasciando alla libreria di I/O (stdio.h) il compito di inserire i caratteri giusti a seconda del S.O. su cui il programma è eseguito
- ciò vale sia in output sia in input

Problema: saluta

Fare un programma che:

- **scriva in output un saluto**

Sottoproblema:

- **come si produce un testo in output?**

Come si produce in output un risultato?

- **si usa la funzione printf**
- **il primo argomento è la "stringa di formato" che contiene:**
  - **i caratteri da scrivere direttamente**
  - **le "direttive di conversione" per indicare il formato da usare per stampare dei valori**
- **gli altri argomenti sono i valori che si vogliono stampare (ad esempio una variabile o un'espressione)**
- **attenzione: ad ogni direttiva di conversione DEVE corrispondere un valore altrimenti l'output generato è impredicibile**
- **sintassi:**

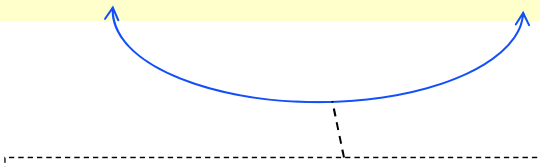
```
int printf ("formato_di_stampa", variabili);
```

Direttive di conversione per la printf

- **%d, %i (int, short, char interpretato come numero intero decimale)**
- **%ld (long)**
- **%u (unsigned)**
- **%lu (unsigned long)**
- **%f (float)**
- **%lf (double)**
- **%c (char, interpretato come carattere ASCII)**
- **%s (char[ ] = stringa di caratteri ASCII)**
- **%x, %X (numero intero interpretato come esadecimale)**
- **%o (numero intero interpretato come ottale)**
- **%% genera il carattere '%'**

Corrispondenza direttive : valori

```
printf ("totale = %d – arrivederci\n", tot);
```



tot deve essere una variabile di tipo intero
+
il suo valore verrà presentato in output come numero decimale

La libreria stdio

- **per poter usare le funzioni di I/O standard del linguaggio C – come printf( ) e getchar( ) – occorre informare il sistema di traduzione affinché:**
  - **il compilatore ne capisca la sintassi**
  - **il linker agganci le opportune librerie**
- **ciò viene fatto grazie alla seguente istruzione, da scrivere in testa al file (N.B. il carattere # deve essere in prima colonna):**

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

Soluzione: saluta (v1, v2)

```
/* saluta (v1) */
#include <stdio.h>

int main ()
{
  printf ("Buona giornata!");
  return 0;
}
```

```
/* saluta (v2) */
#include <stdio.h>

int main ()
{
  printf ("Buona giornata!\n");
  return 0;
}
```

Soluzione: saluta (v3, v4)

```
/* saluta (v3) */
#include <stdio.h>

int main ()
{
  printf ("Buona ");
  printf ("giornata!\n");
  return 0;
}
```

```
/* saluta (v4) */
#include <stdio.h>

int main ()
{
  printf ("Buona ");
  printf ("giornata!");
  printf ("\n");
  return 0;
}
```

Esempio – stampa anno

Scrivere un programma che:

- **memorizzi in una cella di memoria l'anno attuale**
- **indichi in output di quale anno si tratta**

```
#include <stdio.h>

int main ()
{
  unsigned anno = 2009;

  printf ("Buon %u!\n", anno);

  return 0;
}
```

anno.c

Assegnazioni

- per memorizzare un dato in una cella di memoria si usa l'operatore =
- a sinistra del simbolo si indica la cella destinazione
- a destra del simbolo si scrive l'espressione che genera il valore da memorizzare, terminata con ;
- sintassi:

```
variabile = espressione ;
```

- esempi:

```
a = 5.3;
b = 2 * 7;
c = a + b / 2;
d = c;
```

Operazioni aritmetiche

- operatori aritmetici:
  - somma (+)
  - sottrazione (–)
  - moltiplicazione (*)
  - divisione (/)
  - modulo (%) (=resto della divisione intera)
- l'operazione di divisione eseguita tra numeri interi fornisce un risultato intero, ossia il quoziente (scartando il resto)
  - esempio di divisione su operandi interi: 10 / 4 = 2
  - esempio di divisione su operandi float: 10.0 / 4 = 2.5
- l'operazione di modulo è possibile solo tra variabili intere

Espressioni aritmetiche

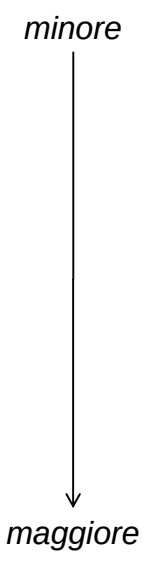
- **le operazioni seguono le normali regole di precedenza**
- **si possono usare le parentesi tonde per alterare l'ordine di precedenza**

`2 + 3 * 7` **è diverso da** `(2 + 3) * 7`

Rango dei tipi numerici

- **le operazioni avvengono dopo aver trasformato tutti gli operandi al tipo di "rango" più alto presente nell'espressione**
- **gli operandi includono tutti gli elementi dell'espressione:**
  - **costanti**
  - **variabili**
  - **risultati di funzioni**
- **unica eccezione: tutti i calcoli che coinvolgono almeno un numero reale (non importa se float o double) avvengono trasformando prima tutti gli operandi in double**

Rango dei tipi numerici




```
_Bool
char
short
unsigned short
int
unsigned int
long
unsigned long
long long
unsigned long long
float
double
long double
```

Rango dei tipi numerici

- **prestare attenzione quando l'istruzione coinvolge tipi interi e reali**
- **esempio di comportamento inaspettato:**

```
float num;

num = 7 / 2;      /* num = 3 */

num = 7 / 2.0;    /* num = 3.5 */
```

char: intero o carattere?

- **le variabili di tipo char possono essere considerate indifferentemente come caratteri ASCII o come numeri interi**
- **l'interpretazione dipende dall'uso che ne fa il programma (ad esempio con un formato "%c" o "%d" in un'operazione di output)**

char.c

Input di dati

- **è una delle fasi più complesse e più soggette ad errori perché comporta un'interazione con un essere umano**
- **… il quale non sempre si comporta come il programmatore si aspetta**
- **il C rende disponibili varie funzioni per leggere dati da input:**
  - **input a caratteri (getchar, gets, …)**
  - **input di dati generici (scanf, …)**

La funzione SCANF è una delle più "pericolose".

Viene qui introdotta solo per permettere di svolgere alcuni esercizi prima di aver imparato il modo corretto per acquisire dati da input.

L'uso di SCANF in un tema di esame comporta una valutazione pesantemente negativa.

Input tramite scanf ( )

- **legge caratteri da input cercando di identificare e convertire i dati specificati nella direttiva di conversione**
- **i dati letti vengono memorizzati nelle variabili indicate come parametri:**
  - **i nomi delle variabili devono essere preceduti da &**
  - **occorre una variabile per ogni direttiva (come in printf)**
  - **stesse direttive della printf (es. %d, %f)**
- **la funzione ritorna il numero di dati identificati e convertiti con successo**

- **prototipo:**

```
int scanf ( "direttive_di_conversione",
  &var1, &var2, ... );
```

Direttive di conversione (scanf)

- **in generale si usano le stesse direttive che si usano come formato di stampa nella funzione printf (%d, %c, %s, …)**
  - **es. “%d %f” legge e memorizza un numero intero ed uno reale**
- **con la specifica %*… si indica che il dato deve essere “saltato” ossia letto ma non memorizzato; non occorre quindi indicare una variabile per esso**
  - **es. “%d %*d %d” legge tre numeri interi ma ne memorizza solo due (il primo ed il terzo)**
- **un numero prima del tipo di dati da convertire indica che il dato occupa esattamente quel numero di caratteri nella stringa di input**
  - **es. "%4d%2d%2d" per leggere i campi della data "20071107" e memorizzarli in tre variabili (anno, mese, giorno)**

Indirizzi delle variabili

- **in C quando si scrive il nome di una variabile si indica il valore memorizzato in essa**
- **eccezioni:**
  - **quando la variabile compare come termine sinistro di un'assegnazione (es. a = …)**
  - **quando la variabile è preceduta dal simbolo &**
- **in questi casi si intende l'indirizzo della cella di memoria corrispondente alla variabile**
- **eccezione: i nomi delle variabili vettoriali (come le stringhe) indicano sempre l'indirizzo della variabile**
- **quindi nella scanf i nomi delle variabili devono essere tutti preceduti da & (a meno che si tratti di stringhe) perché non indicano il valore della variabile ma la cella di memoria ove depositare il valore letto**

Problema (area di un pavimento piastrellato)

- **scrivere un programma che:**
  - **richieda ed acquisisca il lato di una piastrella quadrata (espresso in cm)**
  - **richieda ed acquisisca il numero di piastrelle presenti sul pavimento**
  - **calcoli l'area del pavimento (espressa in metri quadri)**
- **algoritmo risolutivo (pseudo-linguaggio):**
  - **leggere lato (lato) e numero di piastrelle (n_piastrelle)**
  - **area = n_piastrelle * lato * lato**
  - **presentare area in metri quadri (area/10000)**

Soluzione (area di un pavimento piastrellato)

```
#include <stdio.h>

int main ()
{
  unsigned n_piastrelle;
  float lato;
  float area;

  printf ("Lato piastrella [cm]? ");
  scanf ("%f", &lato);
  printf ("Numero piastrelle? ");
  scanf ("%u", &n_piastrelle);
  area = n_piastrelle * lato * lato;
  printf ("Area del pavimento = %f mq\n",
    area/10000);
  return 0;
}
```

pavimento.c

Esempio: problemi usando scanf

Chi volesse rendersi conto dei problemi generati dall'uso di scanf, esegua il programma "pavimento" fornendo i seguenti input:

Lato piastrella [cm]? venti
Numero piastrelle? cento

Lato piastrella [cm]? 20
Numero piastrelle? cento

Lato piastrella [cm]? 20 cm
Numero piastrelle? 100

Lato piastrella [cm]? 7,5
Numero piastrelle? 100

Lato piastrella [cm]? 2 0
Numero piastrelle? 100

Problema (classificazione di un numero)

- scrivere un programma che:
  - richieda ed acquisisca un numero intero
  - indichi se il numero è positivo o negativo
- nota: ai fini del presente problema, lo zero si considera come un numero positivo

- sottoproblema:
  - come si può sapere se un numero è positivo?

Come determinare se una variabile ha un certo valore?

- si usa l'istruzione if
- se la condizione è vera, si esegue l'azione specificata
- se la condizione è falsa, si esegue l'azione specificata dopo la clausola else (se è presente)
- implementa il blocco IF-THEN-ELSE (o IF-THEN) della programmazione strutturata
- sintassi:

```
if ( condizione_Booleana )
  azione_da_fare_se_la_condizione_e'_vera

if ( condizione_Booleana )
  azione_da_fare_se_la_condizione_e'_vera
else
 azione_da_fare_se_la_condizione_e'_falsa
```

Operatori relazionali

- **operatori che:**
  - **effettuano un confronto tra due dati omogenei di tipo base**
  - **producono un valore Booleano**
- **operatori:**
  - **uguale (==)**
  - **diverso (!=)**
  - **maggiore (>)**
  - **maggiore o uguale (>=)**
  - **minore (<)**
  - **minore o uguale (<=)**

Classificazione di un numero: soluzione

```
#include <stdio.h>

int main ()
{
  int numero;

  printf ("Numero? ");
  scanf ("%d", &numero);

  if (numero >= 0)
    printf ("%d e' un numero positivo\n", numero);
  else
    printf ("%d e' un numero negativo\n", numero);

  return 0;
}
```

numero.c

Blocchi di istruzioni

- **un gruppo di istruzioni può essere racchiuso tra parentesi graffe per costituire un blocco**
- **tipicamente usato quando – a seguito del verificarsi di una condizione (es. if, while) – bisogna eseguire più di un'istruzione**
- **esempio:**

```
/* sconto 20% solo se costo maggiore 10 Euro */

if (prezzo > 10)
{
  sconto = prezzo * 0.2;
  printf ("prezzo finale = %f\n",prezzo-sconto);
}
else
  printf ("prezzo finale = %f\n",prezzo);
```

blocco

" = " o " == " ?

- **attenzione alla differenza tra:**
  - **l'operatore di assegnazione (=)**
  - **l'operatore relazionale di uguaglianza (==)**

- **A = 5 significa**
  "assegna alla variabile A il valore 5"
  ossia
  "memorizza il valore 5 nella cella di memoria chiamata A"
- **A == 5 significa**
  "calcola il risultato dell'operazione Booleana
  'il valore di A è uguale a 5?' "
  ossia
  "la cella di memoria chiamata A contiene il valore 5?"

Dati Booleani

- **in C non esiste il tipo dati Booleano (esiste in C99)**
- **per questo scopo si usa qualunque tipo di dato intero con la convenzione:**
  - **FALSE se è pari a zero**
  - **TRUE se è diverso da zero**

Operatori Booleani

- **sono operatori che:**
  - **agiscono su dati Booleani**
  - **generano un valore Booleano**
- **operatori:**
  - **AND (&&)**
  - **OR (||)**
  - **EX-OR (^)**
  - **NOT (!)**
- **utili per scrivere espressioni Booleane complesse**
- **esempio:**

```
a && ( b || ( !c ) )   /* a AND (b OR (NOT c)) */
```

Esempio: boole

- **scrivere un programma che:**
  - **riceva in input i valori di tre variabili Booleane (a b c)**
  - **calcoli il valore dell'espressione Booleana**

    a AND (b OR c')

boole.c

Esercizio: funzione "buco"

- **scrivere un programma che:**
  - **riceva in input il valore dell'ascissa**
  - **calcoli il valore della funzione così definita**
    - **(se | x | >= 1) y = 2**
    - **(se | x | < 1) y = x^2 + 1**
- **nota: non si può fare uso della funzione "valore assoluto" presente nella libreria matematica**

Come si visualizza un messaggio di errore?

- **è preferibile mandare i messaggi di errore sull'unità di output dedicata agli errori: stderr**
- **si usa la funzione fprintf che ha la stessa sintassi della funzione printf ma ha come primo parametro l'unità di output su cui operare**
- **sintassi:**

```
int fprintf (unita',
  "formato_di_stampa", variabili);
```

- **esempio:**

```
fprintf (stderr,
  "errore nella lettura dei dati.\n");
```

Perché usare stderr?

- **se un programma genera molto output in cui sono mischiati dati normali e segnalazioni di errore, diventa difficile notare gli errori**
- **meglio quindi isolare gli errori su un flusso di output separato**
- **in Unix, MS-DOS ed in una finestra di comandi di Windows e Mac è possibile indirizzare stderr su un file (e quindi vedere immediatamente se e quali errori si sono verificati) tramite la sintassi:**

  prog 2> U

- **esempio di redirezione completa (stdin, stdout, stderr):**

```
prog.exe <dati.txt >risult.txt 2>errori.txt
```

Come si termina di colpo un programma?

- **si usa la funzione exit che termina l'esecuzione del programma e restituisce il controllo al sistema operativo**
- **la funzione può ricevere come parametro un codice di uscita (numero intero) da restituire come informazione al sistema operativo**
- **per convenzione un codice di uscita pari a zero indica che il programma è terminato normalmente, mentre si usa un codice diverso da zero nel caso di terminazione dovuta ad un errore**
- **richiede la dichiarazione #include <stdlib.h>**
- **esempi:**

```
exit(0); /* tutto OK */
exit(1); /* errore!!! */
```

Exit o return?

- **al posto della funzione**
  exit (*codice_di_uscita*)
  si può usare l'istruzione
  return *codice_di_uscita* ;
- **nel main, hanno significato equivalente ma spesso si preferisce exit per la terminazione anomala (es. bug) mentre si usa return per un terminazione normale del programma**
- **nei sottoprogrammi, exit e return hanno effetto molto diverso e per buona programmazione sarebbe sempre meglio usare return e lasciare al chiamante la decisione se terminare il programma o meno**

Programmazione "difensiva"

- il test sullo stato "inesistente" nel programma maiuapic.c è un esempio di programmazione difensiva
- bisogna sempre cercare di scrivere programmi che:
  - tollerino e segnalino errori di input
  - controllino anche ciò che può apparire scontato (es. valori di variabili che teoricamente non dovrebbero mai essere possibili)

It's so hard to make fool-proof programs because fools are so ingenious!

Problema: operazioni

Fare un programma che:

- legga dall'input un'operazioni aritmetica nella forma
  operando operazione operando
  senza spazi tra gli operandi e l'operatore
- esegua l'operazione richiesta e ne stampi il risultato
- operatori validi:
  - addizione (+)
  - sottrazione (–)
  - moltiplicazione (* , x, X)
  - divisione (/ , :)
  - modulo (%)

switch

- **l'istruzione switch si può usare in sostituzione di una cascata di if ... else quando i test sono tutti:**
  - **condotti sulla stessa variabile o espressione**
  - **su numeri interi (o equivalenti, come nel caso di variabili di tipo enum o caratteri)**
  - **di uguaglianza**
  - **con condizioni mutuamente esclusive**
- **tramite l'istruzione case si indicano i valori con cui effettuare il confronto**
- **tramite l'istruzione break si "esce" dallo switch**
- **con l'istruzione default (opzionale ma fortemente consigliata) si indicano le istruzioni da fare quando non si è caduti in nessuno dei casi indicati**

Sintassi dell'istruzione switch

```
switch ( espressione )
{
case valore-X :
  /* istruzioni nel caso espressione == X */
  . . .
  break;
case valore-Y :
case valore-Z :
  /* istruzioni nel caso espressione == Y
  oppure espressione == Z */
  . . .
  break;
default:
  /* istruzioni per tutti gli altri casi */
  . . .
}
```

Soluzione: operazioni

- **si leggono i dati (A, OP, B) tramite scanf**
- **si esegue l'operazione, tramite switch (oppure cascata di if-else), e si genera il risultato R**
- **si presenta il risultato (A OP B = R)**

op_sw.c
op_if.c

Problema (conteggio caratteri)

Fare un programma che:

- **legga caratteri**
- **finché ce ne sono in input**
- **e produca in output il numero di caratteri introdotti**

Sottoproblemi:

- **come si legge un carattere?**
- **come si capisce quando non vengono più introdotti caratteri?**
- **come si ripete un'azione?**
- **come si tiene il conto dei caratteri introdotti?**

Come si legge un carattere?

- **si potrebbe usare scanf con formato %c ma è inefficiente**
- **si preferisce quindi usare la funzione getchar ( )**
- **quando viene eseguita restituisce il codice ASCII del carattere fornito in input, ossia un numero 0 ... 255**
- **quando il flusso di dati in input è terminato (oppure si è verificato un errore), restituisce EOF**
- **quindi restituisce un numero intero: è sbagliato (e provoca errori) memorizzare il risultato in una variabile di tipo char**

- **prototipo:**

```
int getchar(void);
```

EOF (End-Of-File)

- **indica che non ci sono più dati in input**
- **il modo in cui si produce un EOF dipende da molti fattori**
- **esempi:**
  - **automaticamente, se si sta leggendo un file**
  - **battere ^Z per indicare EOF in Windows, MS-DOS o VMS**
  - **battere ^D per indicare EOF in Unix, Linux, Mac-OS**

- **N.B. il simbolo ^ indica di tenere premuto il tasto Control (es. CTRL, CNTL) e quindi premere anche il tasto corrispondente alla lettera indicata**

Come si ripete un'azione?

- **si usa l'istruzione while**
- **se la condizione è vera, allora viene svolta l'azione e poi si ri-verifica se la condizione è vera per eventualmente ripetere l'azione**
- **se la condizione è falsa, l'azione non verrà mai svolta**
- **corrisponde esattamente al costrutto WHILE-DO dei flow-chart della programmazione strutturata**
- **sintassi:**

```
while ( condizione_Booleana )
  azione_da_ripetere
```

Come si tiene il conto di una quantità?

- **si usa una variabile (intera) con funzione di "accumulatore" o "contatore"**
- **la si azzera all'inizio**
- **la si incrementa come necessario**

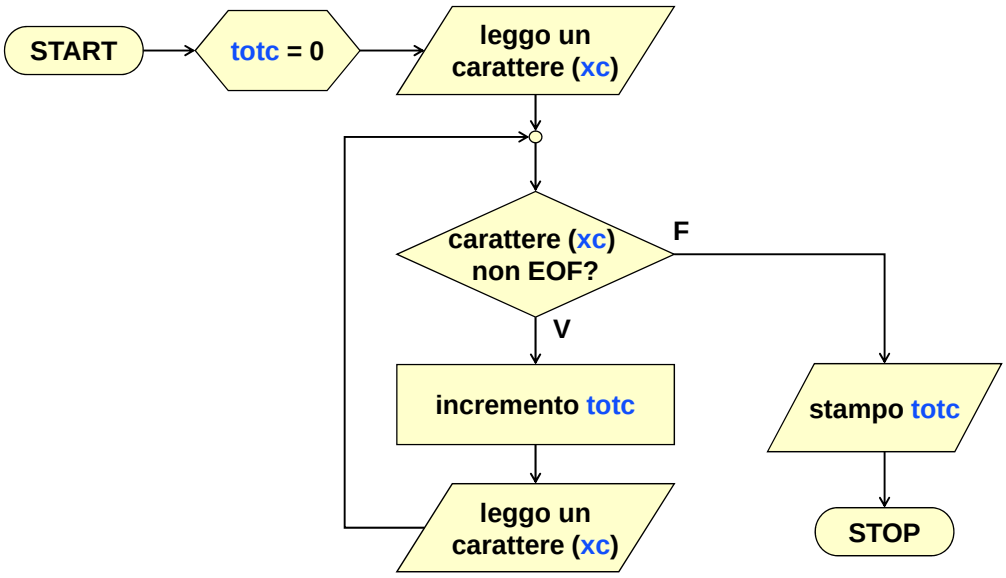
- **esempio:**

```
int tot = 0;

tot++; /* incremento di un'unita' */

tot += 3; /* incremento di tre unita' */
```

Soluzione (conteggio caratteri)

```
int totc = 0; /* totale caratteri */
int xc; /* carattere letto */

xc = getchar();
while ( xc != EOF )
{
  totc++;
  xc = getchar();
}
printf ("Caratteri trovati: %d\n", totc);
```

Flow-chart (conteggio caratteri)



Ottimizzazione (conteggio caratteri)

- **in C si può sostituire ad un valore un'espressione (tra parentesi tonde) che generi tale valore**
- **le parentesi indicano l'ordine di calcolo**
- **poiché nel flow-chart precedente entrambi i rami in ingresso al test svolgono la stessa operazione (lettura di un carattere) essa può essere incorporata nel test stesso, quindi il ciclo while della soluzione può essere così ottimizzato:**

```
while ( (xc = getchar()) != EOF )
{
  totc++;
}
```

contaC.c

Pattern – lettura di tutti i caratteri in input

Ogni volta che occorre leggere tutti i caratteri in input - uno per volta - si usa la seguente struttura base:

```
int xc; /* carattere letto */
...
while ( (xc = getchar()) != EOF )
{
  /*
  inserire qui le operazioni da
  svolgere sul carattere letto (xc)
  */
  ...
}
...
```

Problema (copia caratteri da input a output)

Fare un programma (chiamato "copiac") che:

- **legga caratteri**
- **finché ce ne sono in input**
- **e li copi in output**

Sottoproblemi:

- **come si produce un carattere in output?**

Come si produce un carattere in output?

- **si potrebbe fare con la direttiva %c nella funzione printf ma è inefficiente**
- **meglio usare la funzione putchar ( )**
- **il parametro è il carattere che si desidera produrre in output**
- **la funzione restituisce EOF in caso di errore**

- **prototipo:**

```
int putchar (int carattere);
```

Soluzione (copia caratteri)

```
int xc; /* carattere letto */

while ( (xc = getchar()) != EOF )
{
  putchar(xc);
}
```

copiac.c

Applicazione del programma “copiac”

Il programma copiac può essere usato per visualizzare un file, per generarlo o per copiarlo su un altro file:

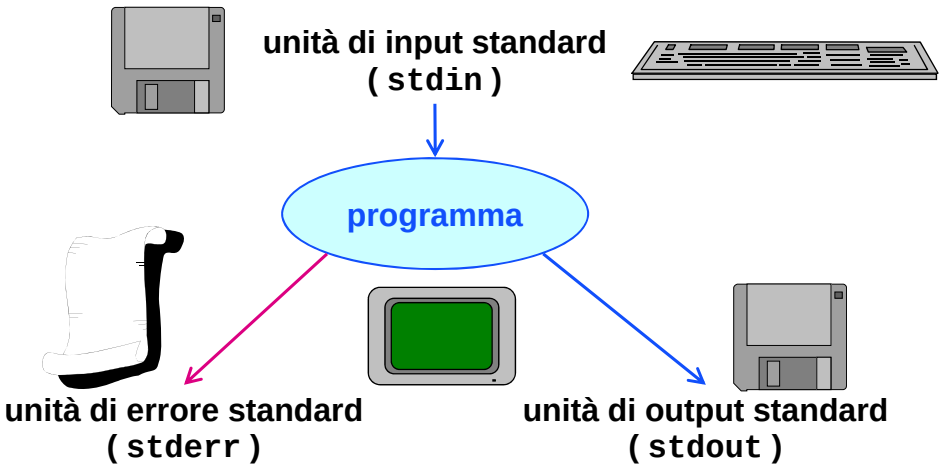
- **(visualizza) copiac < a.txt**
- **(crea) copiac > a.txt**
- **(copia) copiac < a.txt > b.txt**

Nota 1: questi comandi funzionano in Unix, in MS-DOS o in una finestra di comando di MS-Windows o MacOS.

Nota 2: la visualizzazione e la creazione funzionano meglio se si opera con un file di testo (ossia contenente solo caratteri ASCII).

I canali di I/O standard in C

- **il linguaggio C ipotizza l'uso di un calcolatore virtuale dotato di tre canali standard di I/O**



Ridirezione dell'I/O

- **in Unix, MS-DOS ed in una finestra di comando di Windows è possibile cambiare le unità standard di I/O a cui un programma fa riferimento**
- **prog > U**
  dirige l'output sull'unità U
  - **se U contiene già dei dati, essi vengono cancellati**
  - **se U non esiste (es. file) allora viene creata**
- **prog < U**
  prende l'input dall'unità U
- **prog >> U**
  appende l'output (ossia lo aggiunge in coda) all'unità U

Esercizio (conteggio lettera Z)

Fare un programma che conti quante volte compare in input la lettera Z (maiuscola).

Riformulazione del problema = fare un programma che:

- **legga caratteri**
- **finché ce ne sono in input**
- **conti il numero di occorrenze della lettera Z**
- **produca in output il numero di occorrenze della lettera Z**

Esercizio (conteggio lettere Z)

Fare un programma che conti quante volte compare in input la lettera Z (maiuscola o minuscola).

Riformulazione del problema = fare un programma che:

- **legga caratteri**
- **finché ce ne sono in input**
- **conti il numero di occorrenze delle lettere Z e z**
- **produca in output il numero di occorrenze delle lettere Z e z**

Come si ripete un'azione?

- **si può usare l'istruzione do … while (che implementa un ciclo repeat-until)**
- **viene svolta l'azione e poi si verifica se la condizione è vera per eventualmente ripetere l'azione**
- **l'azione verrà svolta almeno una volta**
- **corrisponde al costrutto REPEAT-UNTIL dei flow-chart della programmazione strutturata, avendo però la condizione di uscita invertita**
- **sintassi:**

```
do
  azione_da_ripetere
while ( condizione_Booleana )
```

Problema: leggere una risposta

Chiedere la risposta ad una domanda e ripetere la domanda se la risposta non ha uno dei valori possibili.

Soluzione (astratta e non strutturata):

1. **fare la domanda**
2. **leggere la risposta**
3. **se la risposta non è una di quelle possibili, ripetere da 1**
4. **fornire un commento alla risposta**

Leggere una risposta: soluzione (while)

Usando un ciclo while bisogna duplicare i passi 1 e 2:

- **fare la domanda**
- **leggere la risposta (R)**
- **while (R è diversa da una delle risposte valide)**
  - **fare la domanda**
  - **leggere la risposta (R)**
- **fornire un commento alla risposta**

Leggere una risposta: soluzione (do…while)

Usando un ciclo do-while non si duplica alcun passo:

- **do**
  - **fare la domanda**
  - **leggere la risposta (R)**
- **while (R è diversa da una delle risposte valide)**
- **fornire un commento alla risposta**

rispDo.c

Problema: somma numeri

Sommare tutti i numeri interi positivi forniti in input, terminando la lettura quando viene introdotto un numero non positivo.

Presentare quindi in output la somma di tutti i numeri introdotti.

Somma numeri: soluzione 1 (while-do)

- **totale = 0**
- **leggere un numero (n)**
- **while (n > 0)**
  - **totale = totale + n**
  - **leggere un numero (n)**
- **stampare totale**

Somma numeri: soluzione 2 (do-while)

- **totale = 0**
- **do**
  - **leggere un numero (n)**
  - **totale = totale + n**
- **while (n > 0)**
- **// correzione per ultimo dato aggiunto erroneamente totale = totale – n**
- **stampare totale**

Problema: MCD

Richiedere e leggere due numeri interi e quindi calcolarne il Massimo Comun Divisore (MCD).

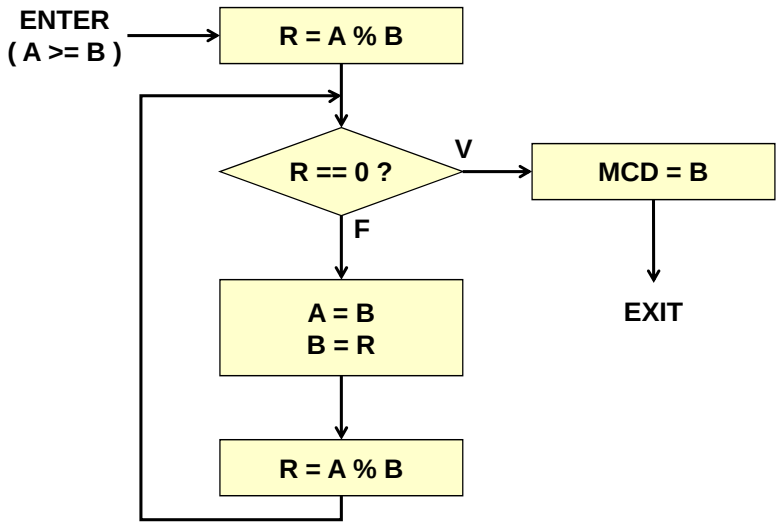
Sottoproblemi:

- **come si calcola il MCD di due numeri interi A e B?**

Algoritmo di Euclide per il calcolo del MCD

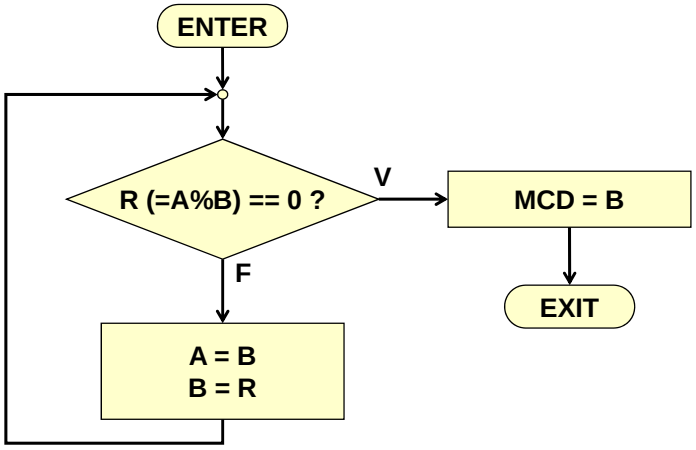
- **algoritmo di Euclide per trovare il MCD di due numeri interi positivi A e B:**
  - **sia A>0, B>0 ed A >= B**
  - **si effettui la divisione intera A / B**
  - **se il resto R della divisione è nullo, allora B è il MCD**
  - **altrimenti si applica nuovamente il procedimento usando B come dividendo e R come divisore**

Flow-chart dell'algoritmo di Euclide (I)



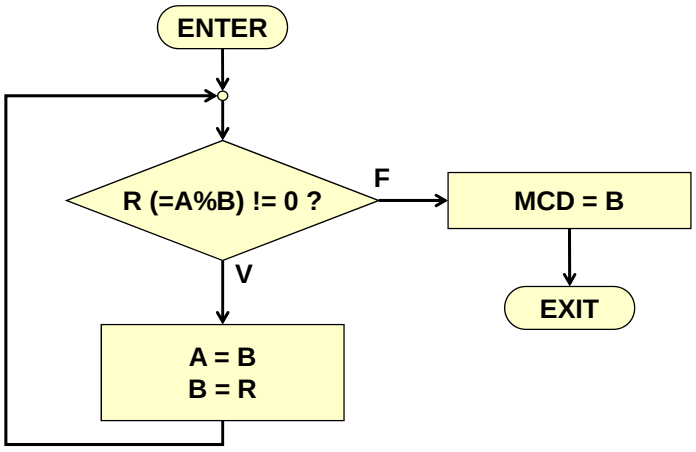
Flow-chart dell'algoritmo di Euclide (II)

- **sfruttando la possibilità del C di combinare calcolo di un valore e suo test …**



Flow-chart dell'algoritmo di Euclide (III)

- **poiché il ciclo while termina quando la condizione è falsa, occorre invertire il test …**



Codifica dell'algoritmo di Euclide

```
/* verifico ed impongo la condizione base a>=b */
if ( a < b )
{
  tmp = a;
  a = b;
  b = tmp;
}
/* calcolo il MCD */
while ( (r = a % b) != 0)
{
  a = b;
  b = r;
}
/* ecco il risultato */
mcd = b;
```

Soluzione: MCD

- **si richiede l'introduzione dei dati**
- **si leggono da input i due numeri interi X e Y tra cui calcolare il MCD**
- **per rispettare la condizione iniziale dell'algoritmo di Euclide (dividendo >= divisore) si pone:**
  - **A pari al maggiore tra X e Y**
  - **B pari al minore tra X e Y**
- **si esegue l'algoritmo di Euclide su A e B**
- **si scrive il risultato**

mcd.c

Problema: tavola dei quadrati (v1)

Stampare la tavola dei quadrati di tutti i numeri interi positivi fino ad un valore massimo introdotto dall'utente.

Sottoproblema:

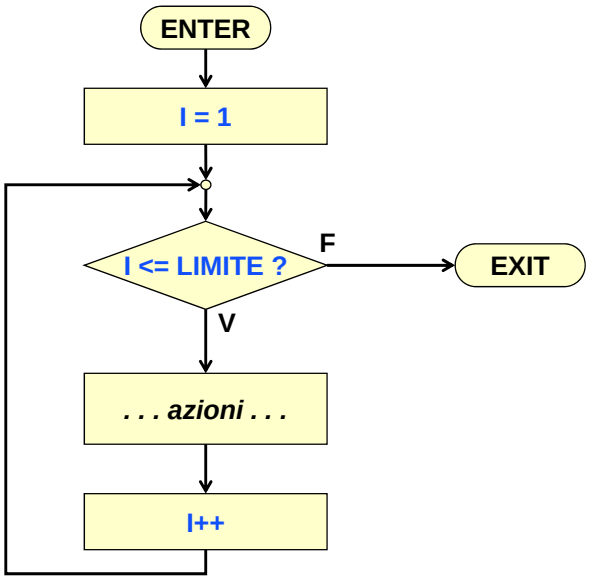
- **come effettuare un ciclo che generi la sequenza degli interi fino ad un certo valore?**

Come generare tutti gli interi fino a N?

- **si usa un "contatore"**
  - **inizializzato al valore 1**
  - **ad ogni iterazione (while) si controlla che sia minore del limite**
  - **al termine di ogni iterazione gli si aggiunge 1**
- **esempio:**

```
int i, limite;
. . .
i = 1;
while (i <= limite) {
  /* azioni per il ciclo i-esimo */
  . . .
  i++;
}
```

Ciclo numerico / contatore: flow chart



Il ciclo " for "

- l'istruzione for è equivalente ad un ciclo while con
  - un'azione preliminare (istruzione da eseguire prima del while)
  - un'azione ciclica (istruzione da eseguire alla fine di ogni passo del ciclo)
- sintassi:

```
for ( azione_preliminare ;
      condizione_Booleana ;
      azione_ciclica )
{
  /* azioni per il ciclo i-esimo */
  . . .
}
```

Pattern – sequenza numerica

Ogni volta che occorre generare una sequenza numerica (limite_inferiore … limite_superiore) si usa una struttura base del seguente tipo:

```
int i, limite_inf, limite_sup;
. . .
for (i=limite_inf; i<=limite_sup; i++)
{
  /* azioni per il ciclo i-esimo */
  . . .
}
```

Incrementi nelle sequenze numeriche

- **nota bene: istruzioni valide solo su variabili intere**

tipo di sequenza	*esempio*
incremento unitario	`i++`
incremento non unitario	`i += 2`
decremento unitario	`i--`
decremento non unitario	`i -= 3`

Soluzione: tavola dei quadrati (v1)

- **richiedo il limite (NMAX)**
- **leggo il limite (scanf) e controllo che sia stato inserito e sia un numero intero positivo (NMAX)**
- **genero tutti gli interi da 1 a NMAX (pattern: sequenza numerica)**
  - **per ogni intero N scrivo il suo quadrato N x N**

tavquad1.c

Problema: tabella Pitagorica

Calcolare e stampare la tabella Pitagorica (ossia la tabella della moltiplicazione tra interi) relativa ai numeri da 1 a N, essendo N un valore introdotto dall'utente.

Soluzione:

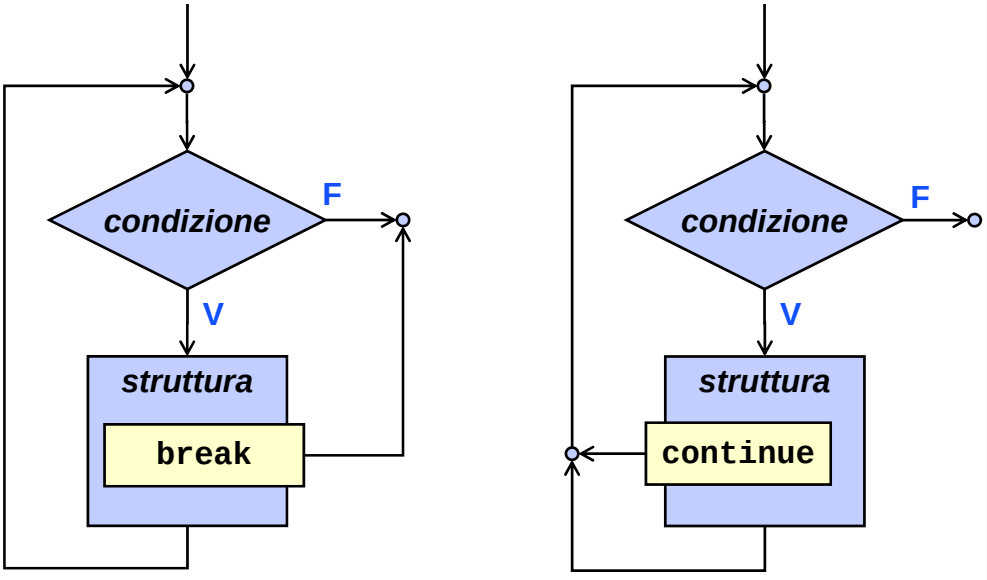
- **chiedo, leggo e controllo il limite della tabella (N)**
- **per ogni riga R da 1 a N … [pattern: ciclo numerico]**
  - **per ogni colonna C da 1 a N … [pattern: ciclo numerico]**
    - **calcolo P = R * C**
    - **stampo P**
  - **vado a capo (riga terminata)**
- **programma terminato**

pitagora.c

Modifica dell'esecuzione dei cicli

- possibile modificare il normale flusso di esecuzione di un qualunque ciclo (while, do-while, for)
- due possibilità:
  - break
    - termina il ciclo
    - esecuzione continua dopo la fine del ciclo
  - continue
    - termina l'iterazione corrente
    - esecuzione continua con la prossima iterazione
- trasformano i cicli in blocchi non strutturati (!)
  - usare con cautela
  - si può sempre evitare l'uso di break/continue … ma talvolta è molto comodo usarli

Esempio break/continue (su while)



Costanti simboliche (const)

- una costante rappresenta un dato non modificabile (ossia una zona di memoria a sola lettura, protetta in tal senso dal sistema operativo)
- è utile assegnare un identificatore mnemonico ad una costante per:
  - ricordare la sua semantica
  - cambiare facilmente il suo valore ovunque sia usata
- sintassi:

  const [*tipo*] *costante* = *valore* ;

- esempi:

```
const float PiGreco = 3.1415;
const int PuntiPerVittoria = 3;
```

Costanti simboliche (#define)

- una costante definita tramite una variabile con attributo “const”:
  - occupa comunque una cella di memoria
  - richiede un trasferimento da memoria a registro per i calcoli (fatte salve possibili ottimizzazioni)
- per non occupare una cella di memoria ma avere ugualmente un identificatore mnemonico associato ad un valore costante si può definire una sostituzione tramite il seguente costrutto:

```
#define identificatore valore_costante
```

- esempio:

```
#define PiGreco 3.1415
#define EOL '\n'
```

Variabili multidimensionali

- **è possibile definire variabili diverse ma correlate che:**
  - **contengono lo stesso tipo di dati**
  - **hanno lo stesso nome (base)**
  - **sono distinte in base ad un indice numerico (prima, seconda, terza, …, N-esima variabile)**
- **queste variabili si chiamano vettori e si dichiarano mettendo dopo il nome una coppia di parentesi quadre che contiene la dimensione del vettore (=numero totale di variabili diverse)**
- **sintassi:**

  ***tipo variabile* [*dimensione*];**
- **esempio:**

```
float punto[3]; // coordinate in 3D
int voto[35]; // voto di ogni studente (max 35)
```

Accesso agli elementi di un vettore

- **non esistono operazioni che si applicano automaticamente a tutti gli elementi di un vettore (a meno che sia una stringa)**
- **normalmente si opera sui singoli elementi del vettore**
- **per accedere ad un singolo elemento di un vettore si specifica il nome del vettore e – tra parentesi quadre – l'indice dell'elemento desiderato**
- **attenzione!**
  - **gli elementi di un vettore con dimensione N sono numerati da 0 a N-1**
  - **quindi l'i-esimo elemento del vettore V si indica con V [i – 1]**

Inizializzazione (statica) di un vettore

- **quando si dichiara un vettore è possibile assegnare valori iniziali ai suoi elementi …**
  - **creando una lista racchiusa tra parentesi graffe**
  - **composta di valori separati da virgola**
- **se un vettore viene inizializzato, è possibile non dichiararne la dimensione, che verrà automaticamente calcolata in modo da poter ospitare tutti i valori specificati (ma nessuno in più)**
- **esempi:**

```
float x[3] = {1.0, 2.1, 3.2};
long y[] = {1, 0, 1, 0};
float z[5] = {1.0, 2.1, 3.2}; // z[3]=0, z[4]=0
int h[2]; // valori casuali
```

Lettura di un vettore (dimensione nota)

```
/* lettura vettore di dimensione nota,
   dati forniti uno per riga su stdin */

#define NDIM 3
. . .
double x[NDIM];
. . .
for (i=0; i<NDIM; i++)
{
  if (scanf("%lf", &x[i]) != 1) {
    fprintf (stderr,
      "errore in lettura elemento n.%d\n", i+1);
    exit(1);
  }
}
```

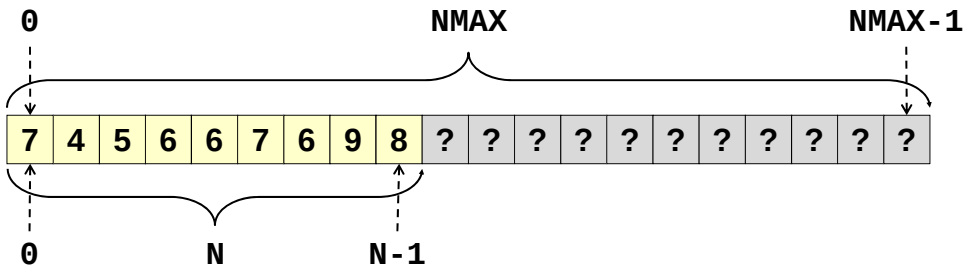
Vettori con dimensione ignota

A priori può essere noto solo il numero massimo di elementi presente ma non quello reale.

Occorre dunque avere due indici:

- **numero massimo di elementi (NMAX)**
- **numero reale di elementi presenti (N)**

Esempio:



Lettura di un vettore (dimensione ignota)

```
/* lettura vettore di dimensione ignota, dati
forniti uno per riga su stdin, fine se negativo */

#define NMAX 100
double x[NMAX];
int nx = 0; // numero di elementi presenti in x
. . .
for (i=0; i<NMAX; i++)
{
  if (scanf("%lf", &x[i]) != 1) {
    fprintf (stderr, "errore in lettura\n”);
    exit(1);
  }
  if (x[i] < 0) break;
}
nx = i;
```

Stampa di un vettore

- **si usa un ciclo numerico su tutti gli indici degli elementi presenti**
- **attenzione ai limiti degli indici: da ZERO a N-1**
- **esempio:**

```
/* stampa vettore (contenente NX elementi) */

double x[...];
int nx; // numero di elementi presenti in x
. . .
for (i=0; i<nx; i++)
{
  printf ("- elemento n.%d: %lf\n", i, x[i])
}
```

Problema: statistiche sui voti

Un programma riceve in input i voti (espressi come numeri interi tra 0 e 10) degli studenti di una classe (massimo 35 studenti).

Il programma deve memorizzare i voti in un vettore e quindi calcolare:

- **il voto minimo**
- **il voto massimo**
- **il voto medio**

Sotto problemi:

- **come calcolare il minimo (massimo) di una serie di valori?**
- **come calcolare la media di una serie di valori?**

Pattern – calcolo del minimo (massimo)

- **si ipotizza che il primo valore letto sia il minimo (massimo) di tutti i valori**
- **ogni volta che si legge un nuovo valore, si corregge l'ipotesi se è sbagliata**

```
double val, minv, maxv;
. . .
/* dopo la lettura di un valore (val) ... */
if ( primo_valore ) {
  minv = val;
  maxv = val;
}
else {
  if (val < minv) minv = val;
  if (val > maxv) maxv = val;
}
```

Pattern – calcolo della media aritmetica

- **il valore medio è la somma di tutti valori diviso il numero di valori**
- **occorre quindi:**
  - **un accumulatore dove sommare tutti i valori**
  - **un contatore per contare il numero di valori**

```
double val, totv = 0;
int nval = 0;
. . .
/* dopo la lettura di un valore (val) ...*/
nval++;
totv += val;

/* alla fine ... */
printf ("media = %lf\n", totv / nval);
```

Soluzione: statistiche sui voti

- **si leggono i valori dei voti**
- **si applicano i pattern per la ricerca di minimo, massimo e media …**
- **… osservando che in un vettore il primo valore è quello con indice zero**

- **NOTA: si possono fare cicli separati per calcolare minimo, massimo e media (vstat1.c) oppure fare un unico ciclo (vstat2.c)**

vstat1.c
vstat2.c

La libreria math

- **contiene funzioni matematiche:**
  - **esponenziali e logaritmiche**
  - **trigonometriche (dirette, inverse, iperboliche)**
  - **miscellanea (valore assoluto, ceiling, …)**
- **attenzione: le funzioni trigonometriche (sia dirette sia inverse) operano su angoli espressi in radianti**
- **per usarla:**

```
#include <math.h>
```

math.h (I)

funzione	*definizione*
`double sin (double x)`	
`double cos (double x)`	
`double tan (double x)`	
`double asin (double x)`	
`double acos (double x)`	
`double atan (double x)`	
`double atan2 (double y, double x)`	atan (y / x)
`double sinh (double x)`	
`double cosh (double x)`	
`double tanh (double x)`	

math.h (II)

funzione	*definizione*
`double pow (double x, double y)`	x^Y
`double sqrt (double x)`	radice quadrata
`double log (double x)`	logaritmo naturale
`double log10 (double x)`	logaritmo decimale
`double exp (double x)`	e^X
`double ldexp (double m, int esp)`	$m\ 2^{esp}$
`double frexp (double x, int *esp)`	mantissa = frexp (x, &esponente)

Nota: ldexp crea un numero reale date la sua mantissa ed esponente, mentre frexp estrae da un numero reale la sua mantissa ed esponente

math.h (III)

funzione	*definizione*
`double ceil (double x)`	ceiling(x)
`double floor (double x)`	floor(x)
`double fabs (double x)`	valore assoluto
`double fmod (double x, double y)`	resto di x / y
`double modf (` `  double x, double *ipart)`	restituisce frac(x) e memorizza int(x) in ipart

testmath.c

Funzioni matematiche in stdlib.h

funzione	*definizione*
`int abs (int n)`	valore assoluto
`long labs (long n)`	valore assoluto
`div_t div (int numer, int denom)`	quoto e resto della divisione intera
`ldiv_t ldiv (long numer, long denom)`	quoto e resto della divisione intera

Nota: div_t e ldiv_t sono struct con due campi (int o long a seconda della funzione usata):

```
quot /* quoto */
rem /* resto */
```

Type cast

- **per trasformare temporaneamente un valore da un tipo ad un altro si può usare un cast**
- **basta premettere alla variabile (o espressione) il tipo desiderato, tra parentesi tonde**
- **esempio: (double) n**
- **utile soprattutto in relazione al passaggio di parametri alle funzioni matematiche**
- **esempio:**

```
int n;
double x;
. . .
x = sqrt ( (double) n );
```

Esercizio

Scrivere un programma che richieda all'utente il valore di un angolo (espresso in gradi, come numero intero) e ne calcoli seno, coseno e tangente.

Nota: se G è un angolo espresso in gradi e R è il suo valore in radianti, allora vale la relazione

$$G : 180 = R : \pi$$

ossia

$$R = G * \pi / 180$$

$$G = R * 180 / \pi$$

trigo.c

Pattern – calcolo della media geometrica

- **la media geometrica è la radice N-esima del prodotto degli N valori**
- **occorre quindi:**
  - **un accumulatore dove moltiplicare tutti i valori**
  - **un contatore per contare il numero di valori**

```
double val, totv = 1;
int nval = 0;
. . .
/* dopo la lettura di un valore ...*/
nval++;
totv *= val;

/* alla fine ... */
printf ("media = %lf\n",pow(totv,1.0/nval);
```

Verifica di proprietà su un insieme di dati

- **universalità – è vero che tutti i dati verificano la proprietà P?**
  - **$\forall$ x : P(x) = true**
- **esistenza – è vero che almeno un dato verifica la proprietà P?**
  - **$\exists$ x : P(x) = true**
- **inesistenza – è vero che nessun dato verifica la proprietà P?**
  - **$\forall$ x : P(x) = false**
- **non universalità – è vero che almeno un dato non verifica la proprietà P?**
  - **$\exists$ x : P(x) = false**

Verifica di proprietà su un insieme di dati

- **per semplicità faremo gli esempi:**
  - **su un vettore v[N]**
  - **ipotizzando che la proprietà cercata sia rappresentata dalla funzione P(x) che restituisce 0 se la proprietà non è soddisfatta, 1 in caso contrario**
- **inoltre assumeremo le seguenti definizioni, utili in generale quando si vogliono trattare dati Booleani senza far ricorso al tipo _Bool, ossia per definire una variabile di tipo FLAG utile per tenere traccia se un certo evento o condizione si è verificata o meno:**

```
#define BOOLEAN int
#define FALSE 0
#define TRUE (!FALSE)
```

Verifica di universalità

- **si controlla se tutti gli elementi soddisfano la proprietà P**

```
BOOLEAN ok = TRUE; /* ipotesi: tutto OK */
. . .
for (i=0; i<N; i++) {
  if ( ! P(vet[i]) ) {
     ok = FALSE;
     break;
  }
}
if (ok)
  printf ("tutti gli elementi soddisfano P\n”);
else
  printf ("almeno un elemento non soddisfa P\n”);
```

Verifica di esistenza

- **si cerca se almeno un elemento soddisfa la proprietà P**

```
BOOLEAN trovato = FALSE;
. . .
for (i=0; i<N; i++) {
  if ( P(vet[i]) ) {
     trovato = TRUE;
     break;
  }
}
if (trovato)
  printf ("esiste un elemento che soddisfa P\n”);
else
  printf ("nessun elemento soddisfa P\n”);
```

Esercizi

Leggere da standard input un vettore di 10 numeri interi e quindi verificare se tutti i numeri sono interi positivi.

Leggere da standard input un vettore di 10 numeri interi e quindi verificare se contiene un numero negativo. In caso affermativo, indicare il suo valore e la posizione del vettore in cui si trova.

Leggere da standard input un vettore di 10 numeri interi e quindi verificare se tutti sono compresi in un intervallo di +/–20% rispetto alla media aritmetica di tutti i valori.

NOTA: l’uso dei vettori per i primi due esercizi è in realtà superfluo (si può anche operare direttamente sui numeri ricevuti in input) mentre è indispensabile per il terzo esercizio.

Sottoprogrammi, procedure e funzioni

- **sottoprogramma (subroutine) = insieme di istruzioni**
  - **a cui viene assegnato un identificatore**
  - **definite una sola volta**
  - **usabili varie volte**
- **funzione (function) = sottoprogramma che restituisce direttamente un risultato**
- **procedura (procedure) = sottoprogramma che non restituisce direttamente un risultato**

- **in C esistono solo le funzioni; le procedure si realizzano con una funzione che restituisce un risultato di tipo void**

Funzioni (definite dall'utente)

- **si deve definire:**
  - **il nome della funzione**
  - **il risultato (tipo)**
  - **i parametri forniti alla funzione (tipo e nome)**
  - **le istruzioni che la compongono**

- **sintassi:**

```
[tipo] nome_funzione ( [parametri_formali] )
{
  /* istruzioni */
  . . .
  return [ valore ] ;
}
```

Note sulle funzioni

- **se una funzione non ha parametri, si usa mettere void tra le parentesi tonde per indicarlo**
- **se una funzione non restituisce esplicitamente alcun risultato (ossia è una procedura) allora:**
  - **lo si indica col tipo void**
  - **si usa return senza alcun valore**
  - **si può anche non usare return (la procedura termina quando finiscono le sue istruzioni)**
- **il main è una particolare funzione: quella che viene attivata per prima alla partenza del programma**
- **nel main si può usare indifferentemente return o exit( )**
- **nelle funzioni:**
  - **return termina solo la funzione**
  - **exit( ) termina tutto il programma**

Posizionamento delle funzioni

- **per semplicità, le funzioni sono definite prima delle parti di programma che le usano**
- **in particolare normalmente tutte le funzioni sono definite prima del main( )**
- **nota : il main() stesso è una funzione**

```
int f1 (int x, int y)
{
/* istruzioni della funzione */
}

int main()
{
/* istruzioni del main */
}
```

Esempio di funzione: volume di un cilindro

Ddefinire una funzione che calcoli il volume di un cilindro dato il raggio della base e l'altezza.

Usare quindi tale funzione per un programma che chieda raggio ed altezza di un cilindro e calcoli il volume.

volcil.c

Problema: triangolo rettangolo

Disegnare in output un triangolo rettangolo con cateti uguali e paralleli ai bordi dello schermo:

- **disegnato tramite un carattere passato come primo parametro sulla linea di comando**
- **con lunghezza del cateto indicata da un numero intero passato come secondo parametro sulla linea di comando**
- **esempio**

```
C:\> trett1 * 5
*
**
***
****
*****
```

Come “disegnare” una figura geometrica?

- **procedere riga per riga; occorre quindi:**
  - **calcolare quante righe occorrono**
  - **procedere col pattern della sequenza numerica (prima riga, seconda riga, …)**
- **per ogni riga stampare i caratteri che occorrono:**
  - **procedere col pattern della sequenza numerica (primo carattere, secondo carattere, …)**

Triangolo rettangolo (soluzione 1)

- **controllare che il numero di parametri sia corretto**
- **acquisire dagli argomenti:**
  - **il carattere da usare per la stampa (CS)**
  - **la lunghezza del cateto (LCATETO)**
- **controllare che la dimensione del triangolo sia compatibile con quella del dispositivo di output (il video in modalità testo è tipicamente 24x80)**
- **(per la stampa si noti che la prima riga deve contenere un solo carattere, la seconda riga due caratteri e così via)**
- **generare LCATETO righe e per ogni riga RIGA**
  - **stampare RIGA volte il carattere CS**
  - **andare a capo**

trett1.c

Triangolo rettangolo (soluzione 2)

- **per chiarezza (e per poterla riutilizzare in altri programmi) può essere utile definire ed usare una funzione che "disegna" una riga con un determinato carattere**

```
void riga_piena (int lriga, char x)
{
  int ncol;

  for (ncol=1; ncol <= lriga; ncol++)
    putchar (x);
  putchar ('\n');
}
```

test_rp.c
test_rp2.c
trett2.c

Come riutilizzare una funzione?

- **volendo usare la funzione riga_piena in vari programmi è possibile procedere in vari modi:**
  - **usare l'editor per copiare il sorgente della funzione all'interno del file sorgente del programma che la vuole usare (come fatto in test_rp.c e trett2.c)**
  - **usare la direttiva #include per includere automaticamente il sorgente della funzione all'interno del file sorgente del programma che la vuole usare (come in test_rp2inc.c)**
  - **creare un file sorgente contenente solo la funzione (lib_rp.c), compilarlo e quindi agganciarlo in fase di link; in questo caso è utile creare un file di header (lib_rp.h) da includersi nel sorgente del programma che vuole usare la funzione (come in test_rp2lib.c)**

lib_rp.c | lib_rp.h | test_rp2inc.c | test_rp2lib.c

Compilazione e link manuale (con gcc)

- **compilazione (genera f1.o) = `gcc -c f1.c`**
- **link (genera prog.exe) = `gcc -o prog.exe f1.o f2.o`**
- **ad esempio, per creare il file eseguibile di test_rp2lib.c si eseguono i seguenti passi:**

```
gcc -c lib_rp.c
gcc -c test_rp2lib.c
gcc -o test_rp2lib.exe test_rp2lib.o lib_rp.o
```

Esercizio: quadrato pieno

- **stampare un quadrato pieno …**
- **… di lato L (passato come primo argomento sulla riga di comando)**
- **… usando il carattere passato come secondo argomento sulla riga di comando**
- **esempio (quadrato di lato 5):**

```
C:\> qpieno 5 *
*****
*****
*****
*****
```

qpieno.c
qpieno2.c

Esercizio: triangolo capovolto

- **stampare un triangolo rettangolo pieno …**
- **… con cateto pari a L (passato come primo argomento sulla riga di comando)**
- **… usando il carattere passato come secondo argomento sulla riga di comando**
- **il triangolo deve essere capovolto (ossia con l'ipotenusa pari alla bisettrice del primo quadrante)**
- **esempio (con cateto = 5):**

```
*****
****
***
**
*
```

Esercizio - quadrato vuoto

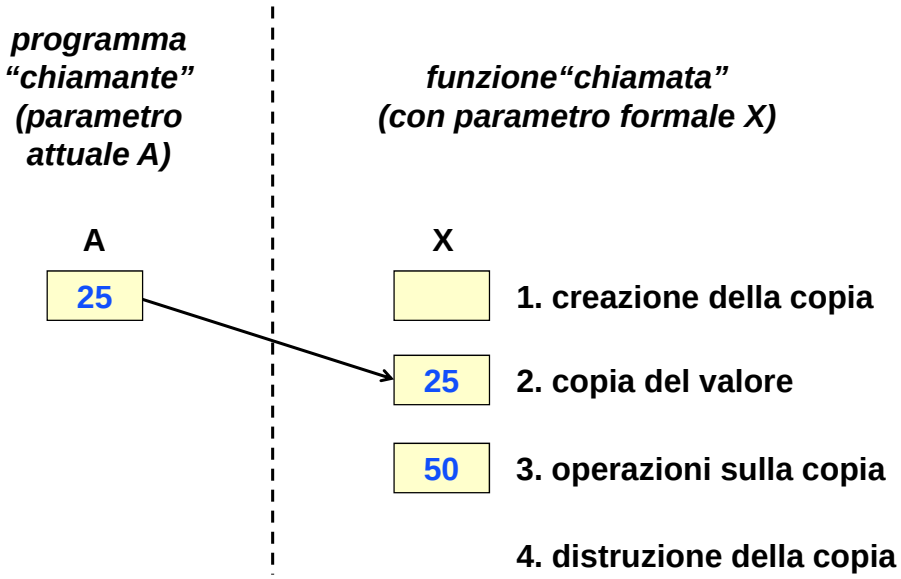
- **stampare un quadrato vuoto …**
- **… di lato L (passato come primo argomento sulla riga di comando)**
- **… usando il carattere passato come secondo argomento sulla riga di comando**
- **esempio (lato = 5):**

```
*****
*   *
*   *
*   *
*****
```

Passaggio dei parametri “by value”

- **nel linguaggio C i parametri alle funzioni sono tutti passati “by value”:**
  - **il valore che si vuole passare viene copiato in una zona di memoria accessibile solo alla funzione**
  - **la funzione opera sulla copia, eventualmente modificandola**
  - **quando la funzione termina, la copia viene distrutta**
  - **nessuna variazione avviene sul valore originale**

Passaggio by value



Esempio 1 – parametri by value

```
void raddoppia (int x)
{
  x = x + x;
  printf ("x = %d\n", x);
}


void main ()
{
  int a = 5;

  printf ("a = %d\n", a);
  raddoppia(a);
  printf ("a = %d\n", a);
}
```

radd1.c

Esempio 2 – parametri by value

```
int raddoppia (int x)
{
  x = x + x;
  return x;
}


void main ()
{
  int a = 5;

  printf ("a = %d\n", a);
  a = raddoppia(a);
  printf ("a = %d\n", a);
}
```

radd2.c

Passaggio dei parametri “by reference”

- **se il risultato di una funzione:**
  - **è uno solo**
  - **è un dato semplice**
- **… allora è possibile restituirlo tramite return( )**
- **se invece:**
  - **i risultati sono tanti e/o sono dati strutturati**
  - **i parametri sono dati strutturati**
- **… allora è necessario agire direttamente sui parametri passando alla funzione non il valore ma un riferimento:**

l’indirizzo di memoria

Indirizzo di memoria

- **l’operatore & premesso al nome di una variabile permette di conoscere l’indirizzo di memoria in cui è conservata tale variabile**
- **eccezione:**
  - **nel caso di variabili strutturate (es. vettori) usando il nome della variabile si ha automaticamente l’effetto di prenderne l’indirizzo**
  - **è quindi superfluo (ma non sbagliato) usare & in fronte di una variabile strutturata**
- **se si vuole memorizzare un indirizzo di memoria, occorre dichiarare una variabile di tipo puntatore a memoria, ossia il cui nome sia preceduto dal simbolo ***

Esempio – indirizzi di memoria

```
#include <stdio.h>

void main()
{
  char alfa = 'a';
  char *aptr = &alfa;

  printf ("alfa = mem(%x) = '%c'\n",&alfa,alfa);
  printf ("aptr = mem(%x) = %x\n", &aptr,aptr);

  *aptr = 'b';

  printf ("alfa = mem(%x) = '%c'\n",&alfa,alfa);
  printf ("aptr = mem(%x) = %x\n",&aptr,aptr);
}
```

eptr.c

Esempio 3 – parametri by reference

```
void raddoppia (int *x)
{
  *x = (*x) + (*x);
}


void main ()
{
  int a = 5;

  printf ("a = %d\n", a);
  raddoppia(&a);
  printf ("a = %d\n", a);
}
```

radd3.c

Esercizio: radtri

- **scrivere una funzione RADTRI che:**
  - **riceve in input un valore intero X**
  - **fornisca in output il doppio ed il triplo del valore X**

radtri.c

Esercizio - conta cifre

Fare un programma che conti (e presenti in output) quante volte compaiono in input delle cifre (ossia dei caratteri numerici).

La libreria ctype

- contiene funzioni per:
  - classificare i caratteri (es. è maiuscolo? è minuscolo? è una cifra?) fornendo una risposta Booleana
  - convertire caratteri da maiuscolo a minuscolo
- per usarla:

```
#include <ctype.h>
```

ctype.h (I)

funzione	*definizione*
`int isalpha(int c);`	`a..z A..Z`
`int isdigit(int c);`	`0..9`
`int isalnum(int c);`	`0..9 a..z A..Z`
`int isxdigit(int c);`	`0..9 a..f A..F`
`int iscntrl(int c);`	`DEL '\0'...'\37'`
`int isprint(int c);`	carattere stampabile
`int isgraph(int c);`	`isprint – isspace`
`int isspace(int c);`	`SP HT CR LF VT FF`
`int ispunct(int c);`	`isprint – isspace - isalnum`

ctype.h (II)

funzione	*definizione*
`int islower(int c);`	`a...z`
`int isupper(int c);`	`A...Z`
`int tolower(int c);`	`A...Z -> a...z`
`int toupper(int c);`	`a...z -> A...Z`

nota: le funzioni tolower e toupper restituiscono il carattere inalterato se non è uno di quelli da trasformare.

Conteggio lettere Z: soluzione n.4

In relazione al problema già visto di contare quante volte compare la lettera Z (maiuscola o minuscola) si può ora usare la funzione tolower( ) – oppure toupper( ) – per convertire il carattere letto e quindi effettuare un solo test:

```
if ( tolower(xc) == 'z' )
  nZ++;
```

```
if ( toupper(xc) == 'Z' )
  nZ++;
```

contaZz4.c

Esercizi

- **contare (separatamente) quante virgole, quanti punti, quanti due-punti e quanti punto-e-virgola compaiono in input**
- **[contaCnoEOL] contare quanti caratteri sono forniti in input, escludendo dal conteggio i terminatori di linea**
- **[contaOper] definire una funzione**

  ```
  int isoper (int  c)
  ```

  che restituisce vero o falso se il carattere ricevuto è un operatore aritmetico (`+ - * /`); usare poi tale funzione per scrivere un programma che conti quante operazioni sono presenti in input.

Esercizio - vocali

Fare un programma che legga tutti i caratteri in input e dica quante vocali sono presenti.

Nota: si desidera minimizzare il numero di test effettuati.

Esercizi - no vocali / consonanti

Fare un programma (chiamato “novoc”) che copi tutto l’input sull’output, tranne le vocali.

Fare un programma (chiamato “nocon”) che copi tutto l’input sull’output, tranne le consonanti.

Esercizio - conta righe

Fare un programma che conti il numero di righe fornite in input.

contaR.c

Problema: valor medio di tre numeri

Vengono fornite in input righe che contengono tre numeri reali (qui indicati con N1, N2 e N3).

Fornire in output righe che contengano il valor medio della corrispondente riga di input, nel seguente formato:

media (N1, N2, N3) = *valore_della_media*

Se una riga di input non contiene tre numeri, segnalare l'errore e terminare il programma, a meno che la riga sia completamente vuota (nel qual caso si deve generare una riga vuota anche in output e proseguire).

Sottoproblema:

- **come si leggono numeri da input?**
- **come ci si accorge che una riga è vuota?**

Valor medio di tre numeri – pseudo-codice

```
leggere una riga
while ( riga letta con successo )
{
  if ( riga vuota )
    stdout <- riga vuota
  else
  {
    estrarre dalla riga i tre numeri N1, N2, N3
    if ( numeri estratti con successo )
      stdout <- (N1+N2+N3)/3
    else
      stderr <- errore (non presenti tre numeri)
  }
  leggere una riga
}
```

Valor medio di tre numeri – pseudo-codice (v2)

```
leggere una riga
while ( riga letta con successo )
{
  if ( riga vuota )
    putchar ('\n');
  else
  {
    estrarre dalla riga i tre numeri N1, N2, N3
    if ( numeri estratti con successo )
      printf("media = %lf\n", (N1+N2+N3)/3.0);
    else
      fprintf (stderr, "errore - non tre num\n");
      exit (1);
  }
  leggere una riga
}
```

Come si leggono numeri da input?

- premesso che stdin fornisce solo ed esclusivamente un flusso di caratteri ASCII, esistono varie soluzioni per estrarre da questo flusso i dati
- la soluzione preferibile è quella che:
  - legge tutta una riga di input e la memorizza in una stringa di caratteri (gets, fgets)
  - analizza la stringa di caratteri cercando ed estraendo i dati desiderati (sscanf)
- è fortemente sconsigliato l'uso delle funzioni scanf e fscanf che:
  - non permettono una lettura ordinata per righe
  - presentano molti problemi, soprattutto quando l'input differisce anche di poco da quello atteso

Come si legge una riga di caratteri da input?

- si usa la funzione gets
- la funzione legge caratteri dall'unità di input standard
  - sino a quando incontra il termine della riga (\n)
  - e memorizza i caratteri in una stringa (detta "buffer"), passata come parametro alla funzione
- nota: gets legge il carattere '\n' ma NON lo inserisce nella stringa
- la funzione ritorna la costante predefinita NULL in caso di errore o fine dei dati (EOF)

- prototipo:

```
char *gets ( char *buffer )
```

Stringhe di caratteri

- una "stringa" di caratteri è una sequenza di caratteri terminata dal carattere NUL ossia '\0'
- una variabile di tipo stringa si crea dichiarando il tipo base (char) e – tra parentesi quadrate – la quantità *massima* di caratteri che potrà contenere
- attenzione a lasciare sempre una spazio disponibile per il terminatore
- si può omettere la dimensione solo se si inizializza la stringa con una costante

- esempi:

```
char x[32]; /* stringa <= 31 char */
char y[] = "xyz"; /* stringa <= 3 char */
char z[32] = "xyz"; /* stringa <= 31 char */
```

Memoria occupata dalle stringhe

- **“ALFA” occupa 5 byte: ‘A’ + ‘L’ + ‘F’ + ‘A’ + NUL**
- **“A” occupa 2 byte: ‘A’ + NUL**
- **‘A’ occupa 1 byte perché è un carattere, non una stringa**
- **la stringa nulla "" occupa 1 byte: NUL**

Dimensionamento di una stringa

- **per indicare il numero massimo di caratteri di una stringa si deve usare un valore numerico intero che sia già noto al tempo di compilazione, perché il compilatore deve riservare lo spazio di memoria per la variabile**
- **non si può quindi usare una const (perché è una variabile, anche se con valore fisso, e quindi creata solo al tempo di esecuzione)**
- **si può però definire simbolicamente un valore costante tramite una direttiva #define**
- **esempio:**

```
#define MAXC 100

char s[ MAXC + 1 ];
```

Estrazione di un carattere da una stringa

- **per estrarre un carattere da una stringa è sufficiente far seguire al nome della stringa l'indice del carattere che si vuole estrarre, tra parentesi quadre**
- **attenzione! le posizioni dei caratteri sono numerate partendo da zero**
- **esempi:**
  - **data la stringa char s[ ] = "Ciao!"**
  - **il primo carattere è s[0] ossia 'C'**
  - **il quinto carattere è s[4] ossia '!'**
- **quindi per sapere se una stringa s è vuota (ossia non contiene dei caratteri) basta fare il test**

```
( s[0] == '\0' )
```

Pattern – lettura di tutte le righe in input

Ogni volta che occorre leggere tutti le righe in input si usa la seguente struttura base (adattando il valore di MAXBC):

```
#define MAXBC 127

char buf[MAXBC+1]; /* riga letta */
...
while ( gets(buf) != NULL )
{
  /*
  inserire qui le operazioni da
  svolgere sulla riga letta (buf)
  */
  ...
}
```

Lettura di tutto l'input: getchar o gets?

- **quando il problema richiede di leggere tutti i dati da input, si potrebbe indifferentemente usare il pattern con getchar o quello con gets**
- **conviene però scegliere il pattern con la funzione orientata al tipo di calcolo che si deve poi compiere sui dati letti da input:**
  - **se si deve operare su singoli caratteri (es. metterli tutti in maiuscolo) allora conviene operare con getchar**
  - **se si deve operare su righe o stringhe o estrarre dei dati (es. lettura di dati numerici) allora conviene operare con gets, seguita dall'opportuna sscanf**
- **attenzione: l'uso di gets (o fgets) presuppone di poter definire una lunghezza massima delle righe di input**

Come si estraggono dati da una stringa?

- **si usa la funzione sscanf**
- **la funzione esamina la stringa cercando di identificare e convertire i dati specificati nella direttiva di conversione**
- **i dati letti vengono memorizzati nelle variabili indicate come parametri**
- **la funzione ritorna il numero di conversioni eseguite con successo**

- **prototipo:**

```
int sscanf ( stringa_dati,
  "direttive_di_conversione",
  indirizzi_delle_variabili );
```

Soluzione: valor medio di tre numeri

- **finché ci sono righe in input (gets, buf)**
  - **se la riga è vuota**
    - **si stampa una riga vuota in output**
  - **altrimenti**
    - **tento di estrarre tre numeri dalla riga (sscanf)**
    - **se l'estrazione ha avuto successo**
      - **si calcola e si stampa la media dei tre numeri**
    - **altrimenti**
      - **si segnala errore**
      - **si termina il programma**

media3n.c

Esercizi

- **scrivere un programma che riceva in input tre numeri per riga e scriva in output il totale di ciascuna colonna; righe errate devono essere scartate, indicandone il numero di riga**
- **scrivere un programma che riceva in input un numero per riga ed effettui la somma solo dei numeri positivi; righe che iniziano col punto esclamativo (!) sono dei commenti e devono essere scartate silenziosamente**

tot3col.c | sumpos.c

Esercizi

- **[oper2n] scrivere un programma che legga due numeri interi (sulla stessa riga) e calcoli il risultato delle quattro operazioni aritmetiche su questi due operandi**
- **scrivere un programma che legga una riga contenente un numero intero e lo presenti in output aumentato del 20%**
- **[tot3fil] scrivere un programma che riceva in input righe contenenti il codice di un filiale (1…3) seguito senza spazi dal ricavo di quella filiale e presenti in output il totale dei ricavi di ciascuna filiale (righe errate sono scartate silenziosamente)**

tot3fil.c

Soluzione (oper2n.c)

```
leggere una riga
if ( riga non letta )
  return 0;

estrarre dalla riga i due numeri (a, b)
if ( numeri estratti con successo )
  stdout <- a+b, a-b, a*b, a/b
else
  stderr <- errore (non presenti due numeri)
```

oper2n.c

La libreria string

- **contiene funzioni per la manipolazione di:**
  - **stringhe (= vettori di caratteri terminati da \0)**
  - **aree di memoria (= vettori di byte)**
- **il tipo *size_t* che compare nelle dichiarazioni delle sue funzioni è assimilabile ad *int***
- **per usarla:**

```
#include <string.h>
```

string.h (I)

funzione	*definizione*
`int strlen (char *s)`	lunghezza (escluso \0)
`char *strchr (char *s, int c)`	cerca il carattere *c* partendo dall'inizio
`char *strrchr (char *s, int c)`	cerca il carattere *c* partendo dalla fine
`char *strstr ( char *s1, char *s2)`	cerca la sottostringa *s2* all'interno di *s1*
`int strcmp (char *s1, char *s2)`	confronto alfabetico
`int strncmp (char *s1, char *s2, size_t maxl)`	confronto alfabetico limitato al massimo a *maxl* caratteri

string.h (II)

funzione	*definizione*
`char *strcat ( char *dst, char *src)`	concatena (=append) *src* a *dst*
`char *strncat ( char *dst, char *src, size_t maxl)`	concatena (=append) *src* a *dst* limitandosi a *maxl* caratteri
`char *strcpy ( char *dst, char *src)`	copia *src* in *dst*
`char *strncpy ( char *dst, char *src, size_t maxl)`	copia *src* in *dst* limitandosi a *maxl* caratteri

string.h - note

- le funzioni di ricerca restituiscono:
  - il puntatore alla prima occorrenza del carattere (o sottostringa) se è stato trovato
  - NULL se il carattere (o sottostringa) non è presente nella stringa
- le funzioni di confronto restituiscono:
  - (0) se le stringhe sono uguali
  - (>0) se la prima stringa è “maggiore” della seconda, ossia viene dopo nell’ordine alfabetico
  - (<0) se la prima stringa è “minore” della seconda, ossia viene prima nell’ordine alfabetico
- NOTA: nell’ordine alfabetico le lettere maiuscole vengono prima delle minuscole

string.h - esempi

```
char alfa[ 64 ] = "super";
char beta[ 64 ] = "pippo";

strlen (alfa)
--> 5

strcat (alfa, beta)
--> alfa = "superpippo" beta ="pippo"
strncat (alfa, beta, 8)
--> alfa = "superpip" beta = "pippo"

strcpy (alfa, beta)
--> alfa = "pippo" beta = "pippo"
strncpy (alfa, beta, 3)
--> alfa = "pip" beta = "pippo"
```

La libreria string – esempi

- **fare un programma che stampi in output il numero di caratteri di cui è composta la parola scritta come primo parametro sulla linea di comando**

estrlen.c

- **fare un programma che:**
  - **legga dalla riga di comando due parole e (opzionalmente) un numero massimo di caratteri su cui effettuare il confronto**
  - **dica se le parole sono uguali o diverse**
  - **se le parole sono diverse, le stampi in output in ordine alfabetico crescente**

estrcmp.c

Esercizio: cambio formato data

Si scriva un programma che riceva in input riferimenti temporali (anno, mese, giorno, ore, minuti e secondi) nella forma:

AAAAMMGGhhmmss

e generi in output riferimenti del tipo

hh:mm:ss GG/MM/AAAA

Ogni riga di input contiene un unico riferimento temporale.

Il programma non deve terminare in caso di errore ma segnalarlo in output.

Esempio: 20110513111234 >>> 11:23:34 13/05/2011

Soluzioni: cambio formato data

- **cdata1.c = schema base, con lettura di numeri interi**
- **cdata2.c = (cdata1) + miglioramento del formato di output (dimensione fissa dei campi)**
- **cdata3.c = schema base, con lettura di stringhe invece che numeri interi**
- **cdata4.c = (cdata2) + miglioramento del formato di output (riempimento con zeri)**
- **note:**
  - **la soluzione "migliore" è quella fornita dal programma cdata3.c perché ricopia in output esattamente i vari campi come sono presenti in input … ma ha problemi con input errati**
  - **se però fossero da controllare i valori dei vari campi (es. mese = 1…12) allora sarebbe da preferire la soluzione cdata4**

cdata1.c	cdata2.c	cdata3.c

Compilazione condizionale

- **le direttive #ifdef e #ifndef permettono di condizionare la compilazione di una parte di programma alla circostanza che sia stata fatta o meno una certa definizione (#define)**
- **la parte di programma compilata condizionalmente è tutta quella compresa sino alla direttiva #endif**
- **esempio (per evitare errori dovuti a doppia definizione):**

```
#ifndef FALSE
#define FALSE 0
#define TRUE (!FALSE)
#define BOOLEAN int
#endinf
```

Pattern: i flag

- **un "flag" è una variabile Booleana che viene usata dai programmatori per ricordare se si è verificata una determinata condizione**
- **occorre:**
  - **definirla**
  - **inizializzarla (a vero o falso a seconda di quale sia l'ipotesi base)**
  - **cambiarne il valore quando si verifica la condizione cercata**
  - **eventualmente ri-inizializzarla quando si ricomincia il lavoro**

Problema: pigreco

Calcolare e visualizzare il valore di π con 9 cifre dopo la virgola.

Sottoproblemi:

- **come calcolare il valore di π?**
- **come richiedere una certa dimensione del formato di output?**

Come calcolare il valore di π ?

- **si può sfruttare una relazione trigonometrica inversa, sfruttando la rappresentazione degli angoli in radianti**
- **ad esempio:**
  - **tan(π /4) = 1 e quindi π = 4* atan(1)**

Direttive di conversione in printf

- **sintassi delle direttive:**

  % [*flag*] [*ampiezza*] [*.precisione*] [*size*] *tipo*

- **ampiezza:**
  - **minima dimensione del campo di stampa**
- **precisione:**
  - **(per float/double) numero di cifre frazionarie da stampare**
  - **(per stringhe) massimo numero di caratteri da stampare**
- **size:**
  - **h (short int), l (long int), L (long double)**

Direttive di conversione in printf

- **flag:**
  - **indicati in tabella**
  - **possibile combinare più flag**

flag	*funzione*
-	**giustificazione a sinistra (default: giustificazione a destra)**
+	**per dati numerici, premettere sempre il segno (default: segno solo ai numeri negativi)**
	ai numeri positivi premettere sempre uno spazio
#	**per numeri ottali/esadecimali, premettere 0 / 0x**
#	**per numeri frazionari, inserire sempre la parte frazionarie anche se nulla**
0	**fare padding con 0 (zero) invece che con spazi**

Direttive di conversione in printf

tipo	*variabile*
`i d`	(intero) formato decimale, con segno
`u`	(intero) formato decimale, senza segno
`o`	(intero) formato ottale
`x X`	(intero) formato esadecimale, con lettere alfabetiche minuscole / maiuscole
`c`	(carattere) formato ASCII
`s`	(stringa) sequenza di caratteri ASCII
`f`	(reali) formato frazionario decimale
`e E`	(reali) formato frazionario esponenziale, con e / E
`g G`	(reali) formato frazionario più corto tra f ed e/E

Soluzione: pigreco

- **calcolo il valore di pigreco, tramite atan(1)**
- **stampo il valore di pigreco con 9 cifre decimali tramite un'opportuno formato in printf**

pigreco.c

Problema: equazione di II grado

Si scriva un programma per trovare le radici di un'equazione di secondo grado, considerando tutti i casi possibili.

$$A\,x^2 + B\,x + C = 0$$

Soluzione: equazione di II grado

- **siano A, B e C i coefficienti**
- **se A è nullo (equazione di primo grado):**
  - **se B è nullo**
    - **se C è nullo, equazione indeterminata**
    - **altrimenti, equazione impossibie**
  - **altrimenti $X = -C / B$**
- **altrimenti, se il discriminante D è:**
  - **(nullo) radici reali coincidenti**
    $X_1 = X_2 = -B / 2A$
  - **(positivo) radici reali distinte**
    $X_1 , X_2 = (-B + /- \text{sqrt}(D)) / 2A$
  - **(negativo) radici complesse coniugate**
    Re = $-B / 2A$ Im = $\text{sqrt}(-D) / 2A$

eq2g.c

Dichiarazione di variabili dentro ai blocchi

- **in C è possibile dichiarare variabili e costanti non solo in testa al programma ma anche all'interno di un blocco (= qualunque sequenza di istruzioni racchiusa tra parentesi graffe)**
- **NOTA (vita delle variabili): le variabili dichiarate dentro ad un blocco:**
  - **vengono "create" (=la memoria viene occupata) quando si inizia l'esecuzione del blocco**
  - **sono accessibili solo all'interno del blocco**
  - **vengono "distrutte" (=la memoria viene liberata) quando termina l'esecuzione del blocco**

Esercizio: tavola numerica (v1)

Stampare la tavola dei quadrati, cubi, radici quadrate e radici cubiche di tutti i numeri interi positivi fino ad un valore massimo introdotto dall'utente (non superiore a 1000).

La stampa deve essere ben incolonnata.

I numeri frazionari devono essere stampati con 6 cifre nella parte frazionaria.

tavnum1.c

Problema: tavola dei quadrati (v2)

Si vuole produrre in output la tavola dei quadrati di tutti i numeri compresi tra 1 e N, ove N è un numero indicato sulla riga di comando (ossia subito dopo il nome del programma).

Sottoproblemi:

- **come leggere un dato presente sulla riga di comando?**

Come leggere dati dalla riga di comando?

- **si usano le variabili argc e argv**
- **per usarle devono essere dichiarate come parametri del main**
- **argc indica il numero di stringhe presenti sulla linea di comando, incluso il nome del programma**
- **argv contiene le stringhe vere e proprie:**
  - **argv[0] è il nome del programma**
  - **argv[*n*] è la stringa n-esima dopo il nome del programma**
- **sintassi:**

```
int main (int argc, char *argv[])
```

test2arg.c | args.c

Soluzione: tavola dei quadrati (v2)

- **controllo che ci siano due argomenti sulla linea di comando (argc)**
- **leggo il primo parametro (sscanf, argv[1]), controllo che sia un numero intero e lo memorizzo (IMAX)**
- **genero tutti gli interi sino a IMAX (pattern: sequenza numerica)**
  - **per ogni intero I scrivo il suo quadrato I*I**

tavquad2.c

Esercizi

(r4.c) Scrivere un programma che calcoli la radice quarta di un numero (floating-point) passato sulla riga di comando.

(r4s.c) Scrivere un programma che calcoli la radice quarta di tutti i numeri (floating-point) passati sulla riga di comando.

Scrivere un programma che calcoli la somma di due numeri (floating-point) passati come argomenti sulla riga di comando.

Scrivere un programma che riceva sulla riga di comando due numeri (floating-point) ed un'operazione (+ o –) e calcoli il risultato dell'operazione.

Esercizio (noco)

Scrivere un programma che legga coppie nome e cognome e le memorizzi in due vettori.

La fine delle coppie è segnalata da una riga vuota.

Il programma chiede quindi all'utente un cognome e presenta il nome corrispondente.

Nota. Questo esercizio esemplifica due cose importanti:

- **matrici di caratteri usate come vettori di stringhe**
- **uso di vettori paralleli per conservare informazioni correlate**

Esercizio - intervallo

Scrivere un programma che legga da riga di comando i limiti di un intervallo numerico e quindi stampi in output tutti i numeri interi inclusi nell'intervallo.

Esempio: "interv 3 9" deve generare il seguente output

3
4
5
6
7
8
9

interv.c

Esercizio - scacchiera

Disegnare in output una scacchiera quadrata di lato N (passato come primo argomento sulla linea di comando), con le caselle bianche ottenute stampando uno spazio e le caselle nere ottenute stampando il carattere passato come secondo parametro sulla linea di comando.

scac.c

Problema: copia file in maiuscolo

Copiare il contenuto di un file in un altro file trasformando tutti i caratteri alfabetici in maiuscolo.

I nomi dei due file sono passati come parametri sulla riga di comando.

Sottoproblemi:

- **come accedere ad un file di cui si conosce il nome?**
- **come leggere caratteri da un file?**
- **come scrivere caratteri su un file?**

Come accedere ad un file?

- in C i file sono visti come una sequenza di byte (byte stream): B0 B1 B2 B3 …
- i byte vengono letti o scritti sequenzialmente
- per tenere traccia dello "stato" di un file si deve usare una variabile di tipo FILE * (detta handle)
- per accedere ad un file si usano le seguenti funzioni:
  - fopen( ) per iniziare l'accesso al file
  - fclose( ) per terminare l'accesso al file
  - fgetc( ) / fputc( ) per leggere/scrivere caratteri
  - fgets( ) / fputs( ) per leggere/scrivere stringhe (righe) dal file
  - fprintf( ) per scrivere dati formattati sul file

Apertura di un file

- si usa la funzione fopen che ha come parametri:
  - il nome del file da aprire
  - il tipo di operazioni da compiere (detto "modo")
    "r" = read (lettura)
    "w" = write (scrittura con creazione o cancellazione dei dati già esistenti)
    "a" = append (scrittura con creazione o aggiunta al fondo del file)
- la funzione restituisce:
  - la handle del file, se l'apertura è riuscita
  - il valore NULL se l'apertura è fallita
- sintassi:

```
FILE *fopen ( char *nome_file, char *modo )
```

Pattern: apertura di un file in lettura

- si usa una fopen con parametro read
- se la risposta è NULL si segnala errore

```
FILE *afh;
char fname[] = "prova.txt";
. . .
afh = fopen (fname, "r");
if (afh == NULL)
{
  fprintf (stderr, "errore – non riesco ad
aprire il file '%s' in lettura\n", fname);
  exit (1);
}
. . .
/* faccio accesso al file afh */
. . .
```

Chiusura di un file

- si usa la funzione fclose che ha come parametri:
  - la handle del file da chiudere
- la funzione restituisce:
  - zero se l'operazione è riuscita
  - un numero diverso da zero se la chiusura è fallita
- sintassi:

```
int fclose ( FILE *file_handle )
```

Pattern: chiusura di un file

- si usa una fclose
- se la risposta non è zero, si emette un “warning”

```
FILE *afh;
char fname[] = "prova.txt";
. . .
if (fclose(afh) != 0)
{
  fprintf (stderr,
  "avviso – chiusura del file '%s' non riuscita\n",
  fname);
}
```

Lettura di caratteri da un file

- si usa la funzione fgetc che ha come parametri:
  - la handle del file su cui operare
- la funzione restituisce:
  - il carattere letto, se l’operazione è riuscita
  - EOF se c’è stato errore o si è arrivati alla fine del file
- sintassi:

```
int fgetc ( FILE *file_handle )
```

Scrittura di caratteri su un file

- si usa la funzione fputc che ha come parametri:
  - il carattere da scrivere
  - la handle del file su cui operare
- la funzione restituisce:
  - il carattere scritto, se l'operazione è riuscita
  - EOF se c'è stato errore
- sintassi:

```
int fputc ( int c, FILE *file_handle )
```

Lettura di una stringa (riga) da un file

- si usa la funzione fgets che ha come parametri:
  - la stringa (buf) ove memorizzare i dati letti
  - la dimensione massima della stringa (max), di solito pari alla lunghezza del buffer
  - la handle del file su cui operare
- la funzione memorizza nel buffer al massimo max-1 caratteri letti (incluso il terminatore di riga) ed aggiunge quindi il terminatore di stringa
- la funzione restituisce:
  - NULL se c'è stato errore o si è arrivati alla fine del file (EOF)
- sintassi:

```
char *fgets (
  char *buf, int max, FILE *file_handle )
```

Nota: uso di fgets su stdin

- si noti che la funzione gets non controlla che la dimensione del buffer sia sufficiente a contenere tutti i caratteri letti
- questo controllo viene invece fatto dalla fgets
- può quindi essere utile usare la fgets (invece della gets) anche per la lettura di righe da input invece che da file
- per fare ciò usare come handle di file stdin
- notare che stdin, a differenza delle altre variabili di tipo FILE*, non necessita di essere aperto e chiuso esplicitamente

Scrittura di stringhe (righe) su file

- si usa la funzione fputs che ha come parametri:
  - la stringa (buffer) contenente i dati da scrivere
  - la handle del file su cui operare
- in caso di errore restituisce EOF
- attenzione: fputs( ) non mette automaticamente il ritorno a capo, come invece fa puts( )
  - si può inserire '\n' al fondo della stringa
  - oppure fare fputs(buf,afh) + fputc('\n', afh)
- sintassi:

```
int fputs (
  char *buffer, FILE *file_handle )
```

Scrittura di dati su un file

- **si usa la funzione fprintf che ha la stessa sintassi e funzionalità della printf ma come primo parametro ha la handle del file su cui operare**
- **la funzione restituisce:**
  - **EOF se c'è stato errore**
- **sintassi:**

```
int fprintf (
  FILE *file_handle, formato, variabili )
```

Copia file in maiuscolo (soluzione)

- **controllare il numero di argomenti (argc == 3?)**
- **aprire il primo file (A) in lettura (argv[1], fopen)**
- **aprire il secondo file (B) in scrittura (argv[2], fopen)**
- **finché ci sono caratteri nel file A**
  - **leggere un carattere (fgetc, XC) dal file A**
  - **scrivere il carattere (fputc, XC) sul file B dopo averlo trasformato in maiuscolo (se necessario)**
- **chiudere il file A**
- **chiudere il file B**

maiuf.c

Esercizi

NATALE

Fare un programma che aggiunga la riga “Buon Natale!” in coda ad un file il cui nome è specificato come primo parametro sulla riga di comando

APPENDS

Fare un programma che aggiunga in coda ad un file (il cui nome è specificato come primo parametro sulla riga di comando) una riga contenente la stringa passata come secondo parametro sulla linea di comando

Esercizio: copia N righe

Scrivere un programma che:

- **copi le prime N righe da un file ad un altro**
- **il numero N di righe da copiare è specificato come primo parametro sulla linea di comando**
- **i nomi dei due file su cui operare sono passati come secondo e terzo parametro sulla linea di comando**
- **emetta un avviso nel caso che siano state copiate meno di N righe (a causa di un errore o per mancanza di dati nel file di input)**

Copia N righe (soluzione 1)

- **controllo che ci siano 3 parametri (argc == 4)**
- **controllo che argv[1] sia un numero intero e lo leggo (N)**
- **apro il primo file (argv[2]) in lettura (AFH)**
- **apro il secondo file (argv[3]) in scrittura (BFH)**
- **(contatore) righe_copiate = 0**
- **finché (ci sono righe nel file A) e (righe_copiate < N)**
  - **leggo una riga (BUF) dal file A**
  - **scrivo la riga sul file B**
  - **righe_copiate++**
- **chiudo i file A e B**
- **se (righe_copiate < N) segnalo l'errore**

copiaN1.c

Copia N righe (soluzione 2)

- **la soluzione 1 non funziona correttamente quando una riga è più lunga di MAXBC**
- **una soluzione migliore è quella che copia i caratteri individualmente, in modo da non porre limiti alla lunghezza massima di una riga**
- **conviene quindi scrivere una funzione copia_riga che:**
  - **copia caratteri dal file F1 al file F2**
  - **si arresta dopo aver copiato il carattere \n**
  - **si arresta anche nel caso di EOF**
  - **ritorna 0 in caso di copia terminata a fine riga, EOF se ha letto EOF**

copiaN2.c

Esercizi (totali)

TOT1D

Fare un programma che sommi tutti i numeri interi presenti (uno per riga) nel file specificato come primo argomento sulla riga di comando e presenti il totale in output.

TOT3D

Fare un programma che sommi per colonne tutti i numeri interi presenti (tre per riga) nel file specificato come primo argomento sulla riga di comando e presenti in output il totale di ciascuna colonna.

Il programma deve segnalare le righe con meno di tre dati e non considerarle nel calcolo del totale.

Esercizi (concatenazione di file)

CONCATENAZIONE DI FILE (v1)

Fare un programma che legga il file A ed il file B (secondo e terzo parametro sulla riga di comando) e metta il loro contenuto uno in coda all'altro nel file DEST (primo parametro sulla riga di comando).

Nota: se il file DEST contiene già dei dati, questi devono essere cancellati.

CONCATENAZIONE DI FILE (v2)

Fare un programma che legga tutti i file A, B, C, … (che occupano il secondo, terzo, quarto, … posto sulla riga di comando) e metta il loro contenuto uno in coda all'altro nel file DEST (primo parametro sulla riga di comando)

Esercizio

- **un file contiene su ogni riga i voti di quattro studenti (quattro numeri interi separati da uno spazio), una riga per ogni giorno di scuola**
- **il nome del file è specificato come primo parametro sulla linea di comando**
- **il programma deve produrre in output il voto medio giornaliero di ciascuno dei quattro studenti**
- **l'output deve essere scritto nel file il cui nome è passato come secondo parametro sulla linea di comando**

Esercizio

- **scrivere un programma che legga un file di caratteri (il cui nome è passato come primo argomento sulla riga di comando) e calcoli la dimensione della sua riga più lunga ed indichi il numero d'ordine di tale riga**
- **nota: sviluppare la soluzione nei due casi (A) che non sia nota a priori la massima lunghezza di una riga e (B) ipotizzando che nessuna riga sia più lunga di 1023 caratteri.**

Ricerca caratteri

Fare un programma che:

- **dato un file specificato come primo parametro sulla linea di comando**
- **dato un carattere specificato come secondo parametro sulla linea di comando**
- **visualizzi tutte le righe del file che contengono il carattere dato**
- **stampi il numero di righe che contengono il carattere dato**

Si assuma che ogni riga del file non sia lunga più di 132 caratteri.

Ricerca caratteri – soluzione

- **controllo dei parametri (argc==3, strlen(argv[2])==1)**
- **apertura del file (fopen(argv[1]))**
- **lettura di tutte le righe del file (while + fgets)**
- **per ogni riga, se essa contiene il carattere dato (strchr (buf, argv[2][0]) != NULL):**
  - **stampare la riga (printf)**
  - **incrementare un contatore (NR)**
- **alla fine, stampare il valore del contatore (NR)**

csearch.c

Ricerca caratteri con opzioni

Fare un programma che:

- **dato un file specificato come penultimo parametro sulla linea di comando**
- **dato un carattere specificato come ultimo parametro sulla linea di comando**
- **stampi il numero di righe che contengono il carattere dato**
- **se il primo parametro è "-v" il programma deve anche visualizzare tutte le righe del file che contengono il carattere dato**

Si assuma che ogni riga del file non sia lunga più di 132 caratteri.

Esempi di uso:

- `csearch2 W dati.txt`
- `csearch2 –v W dati.txt`

csearch2.c

Esercizio – quotazioni di borsa

Fare un programma che:

- **legga da un file (il cui nome è fornito come primo parametro sulla linea di comando) le quotazioni in euro di un numero imprecisato di azioni (massimo: 100 azioni) una per riga**
- **fornisca in output la quotazione delle azioni il cui numero d'ordine è fornito in input**

azioni.c

Lavorare su vettori o su file?

Quando dei dati si trovano in un file ha senso caricarli in un vettore se e solo se:

- **sono in quantità nota – o limitata – a priori (altrimenti non si riesce a dimensionare il vettore)**
- **devo usarli più di una volta (altrimenti si perde tempo rispetto a lavorare direttamente sul file)**

Esempi (varianti del problema delle azioni):

- **fornire il valore dell'azione il cui indice è passato come secondo parametro sulla riga di comando [azioni1.c]**
- **non è noto il numero massimo di azioni presenti nel file**

Problema – conversione del mese

Scrivere un programma che riceva in input date nella forma numerica

GG.MM.AAAA

e le trasformi in output nel formato alfabetico

GG *nome-del-mese* AAAA

Come nomi dei mesi si devono usare "Gennaio", "Febbraio", "Marzo", …

Conversione del mese - soluzione “facile”

```
int g, m, a;
char mese[64];

gets (buf);
sscanf (buf, "%d.%d.%d", &g, &m, &a);
switch (m)
{
case 1: strcpy (mese, "Gennaio"); break;
case 2: strcpy (mese, "Febbraio"); break;
...
case 12: strcpy (mese, "Dicembre"); break;
default: strcpy (mese, "(mese sconosciuto)");
}
printf ("%d %s %d\n", g, mese, a);
```

Tabella dei mesi

Può essere utile scrivere una funzione che:

- **riceva come parametro di input il numero di un mese**
- **restituisca una stringa col nome del mese**

Tabella dei mesi

```
char *nome_mese (const int n_mese)
{
  static char *mtab[] = {
    "(mese inesistente)",
    "Gennaio",
    "Febbraio",
    . . .
    "Dicembre"};

  if (n_mese > 0 && n_mese < 13)
    return mtab[n_mese];
  else
    return mtab[0];
}
```

Static?

- **normalmente le variabili vengono:**
  - **create ed inizializzate quando inizia l'esecuzione del blocco che le definisce**
  - **cancellate e distrutte quando termina l'esecuzione del blocco che le definisce**
- **invece le variabili di tipo static vengono:**
  - **create ed inizializzate quando inizia l'esecuzione del programma**
  - **cancellate e distrutte quando termina l'esecuzione del programma**
  - **se sono definite all'interno di una funzione, mantengono il loro valore anche tra chiamate diverse della funzione**

Tabella dei mesi (con calcolo automatico della dimensione)

```
char *nome_mese (const int n_mese)
{
  static char *mtab[] = {
    "(mese inesistente)",
    "Gennaio",
    "Febbraio",
    . . .
    "Dicembre"};
  static int max_mtab =
    sizeof(mtab) / sizeof(mtab[0]);

  if (n_mese > 0 && n_mese < max_mtab)
    return mtab[n_mese];
  else
    return mtab[0];
}
```

sizeof( )

- la funzione sizeof restituisce il numero di byte occupati in memoria da una variabile
- può ricevere come parametro:
  - il nome di una variabile
  - il nome di un tipo di dati

esizeof.c

Conversione del mese – soluzione

- **leggere tutte le righe in input (while + gets)**
- **per ogni riga:**
  - **leggere la data controllandone il formato (sscanf)**
  - **stampare la data in output dopo aver convertito il mese tramite la funzione nome_mese( )**

cmese.c

Pattern: da numero a dato

L'esempio della tabella per la conversione da numero a nome del mese è tipico di tutti i problemi dove bisogna accedere ad un dato (in questo caso una stringa alfabetica) partendo da un numero (un indice).

In generale, per risolvere questi problemi:

- **si crea un vettore, con tipo base pari al tipo dei dati da memorizzare**
- **si inizializza il vettore coi dati**
  - **direttamente in modo statico se i dati sono noti a priori**
  - **oppure dinamicamente (ad esempio leggendo i dati da file)**

Problema – ricavo totale

Nel file PREZZI.TXT sono indicati i prezzi in Euro di venti prodotti.

Sull'unità di input standard vengono fornite righe che contengono due elementi: il numero identificativo del prodotto (compreso tra 1 e 20) e la quantità venduta.

Scrivere un programma che calcoli e presenti in output il ricavo totale.

Ricavo totale (soluzione)

- **pattern: corrispondenza indice – dati:**
  - **usare un vettore che memorizzi il prezzo dei prodotti (PREZZO)**
  - **inizializzare il vettore leggendolo dal file PREZZI.TXT**
- **pattern: leggere tutte le righe in input**
- **per ogni riga letta:**
  - **estrarre i campi (IPROD, QPROD) e verificarne la congruenza**
  - **calcolare il ricavo del prodotto (RIPROD) e sommarlo al totale (RITOT)**
- **alla fine stampare in output il ricavo totale**

ricavo.c

Problema – conversione del mese (con lettura dei nomi da file)

Scrivere un programma che riceva in input date nella forma numerica

GG.MM.AAAA

e le trasformi in output nel formato alfabetico

GG *nome-del-mese* AAAA

Come nomi dei mesi si devono usare i dodici nomi specificati (uno per riga) nel file N.DAT; è garantito che ciascun nome presente in questo file non è più lungo di quindici caratteri.

Al programma può essere passato opzionalmente un parametro sulla linea di comando per richiedere che il nome del mese sia scritto tutto in minuscolo (–L) o in maiuscolo (–U).

Definizione di un vettore di vettori (matrice)

- **per definire un vettore il cui tipo base è a sua volta un vettore (come capita nel caso di vettori di stringhe) è sufficiente usare più volte le parentesi quadre per indicare la dimensione degli indici**
- **un vettore di vettori è anche detto matrice**
- **nel caso di due indici, il primo indica le righe ed il secondo indica le colonne della matrice**

- **esempi:**

```
char nomi_delle_stagioni[4][32];
float matrice2x3[2][3];
```

Conversione di una stringa in maiuscolo

- **si esamina ciascun carattere della stringa e gli si applica la funzione toupper( )**
- **può essere utile definire una volta per tutte la funzione stoupper( ) e la sua gemella stolower( )**

```
void stoupper (char *s)
{
  int i;

  for (i=0; s[i]!='\0'; i++)
    s[i] = toupper( s[i] );
}
```

oppure (più lenta):

```
  ...
  for (i=0; i<strlen(s); i++)
```

Variabili globali

- **è possibile definire variabili anche all'esterno dei blocchi (ossia fuori da qualunque funzione)**
- **queste si chiamano variabili globali e sono accessibili a tutte le funzioni che sono definite dopo di loro nel file sorgente**

Soluzione – conversione del mese (con lettura dei nomi da file)

- pattern: corrispondenza indice – dati:
  - usare un vettore globale che memorizzi i nomi dei mesi (MTAB = 12 stringhe da 15 caratteri)
  - inizializzare il vettore leggendo dal file N.DAT
- lettura e controllo parametri sulla riga di comando
- trasformare in maiuscolo o in minuscolo tutti i nomi dei mesi a seconda dell'opzione (stoupper, stolower)
- leggere tutte le righe in input (while + gets)
- per ogni riga letta:
  - leggere e controllare la data (sscanf)
  - stampare la data in output convertendo il mese tramite la funzione nome_mese( )

cmesef.c

Definizione di nuovi tipi

- in C è possibile creare un nuovo tipo di dati mediante la seguente definizione:

  typedef *tipo identificatore-nuovo-tipo*

- esempi:

```
typedef unsigned char Bit8;

typedef enum {pera, mela, melone} Frutta;
```

Vantaggi della definizione di nuovi tipi

- **il compilatore può controllare che vengano assegnati solo valori coerenti col tipo della variabile**

Le struct

- **in C è possibile definire variabili strutturate composte da:**
  - **elementi eterogenei**
  - **distinti in base al loro nome**
- **questo tipo di dati si chiama struct ed è composta da campi (field)**
- **definizione:**

```
struct [ nome-della-struct ]
{
   tipo-1 nome-campo-1 ;
   tipo-2 nome-campo-2 ;
   . . .
};
```

Esempio: gestione anagrafe studenti

```
typedef enum {gen, feb, mar, …, dic} tipoMese;
typedef enum {1,2,3,4,5,6,…,31} tipoGiorno;
typedef enum {M, F} tipoSesso;

struct personaAnagrafica {
  char nome[32];
  char cognome[32];
  tipoGiorno giorno_di_nascita;
  tipoMese mese_di_nascita;
  unsigned int anno_di_nascita;
  tipoSesso sex;
};

struct personaAnagrafica studente1, studente2;
struct personaAnagrafica corso[MAXSTUDENTI];
```

Esempio: geometria in R^2

```
/* definizione della struttura base */

struct _puntoR2 {
  double x;
  double y;
};

/* definizione tipo per le variabili */

typedef struct _puntoR2 puntoR2;

/* definizione di alcune variabili */

puntoR2 a, b;
puntoR2 origine = {0,0};
```

Accesso ai campi di una struct

- **riguardo ad una struct, è lecito:**
  - **trattarla come un dato unico (assegnarla, confrontarla, passarla come parametro)**
  - **prenderne l'indirizzo**
  - **prendere l'indirizzo di un campo**
  - **accedere ai singoli campi:**
    - **con notazione "`.`" per le variabili**
    - **con notazione "`->`" per i puntatori**
- **esempi:**

```
puntoR2 alfa = {5, 5};
puntoR2 *aptr = &alfa;
alfa.x = 0;
aptr->y = 0;
```

Esempio: calcolo distanza tra due punti in R^2

- **passando direttamente i punti:**

```
double distanzaR2 (puntoR2 p1, puntoR2 p2)
{
  double dx = p1.x - p2.x;
  double dy = p1.y - p2.y;
  return sqrt( dx*dx + dy*dy );
}
```

- **oppure passando i puntatori ai punti:**

```
double distanzaR2 (puntoR2 *p1, puntoR2 *p2)
{
  double dx = p1->x - p2->x;
  double dy = p1->y - p2->y;
  return sqrt( dx*dx + dy*dy );
}
```

Esame del 11-lug-2000

Programma che legga un file di testo e ne produca una copia con le righe "centrate".

Argomenti sulla riga di comando:

- **il nome del file in ingresso**
- **il nome del file di uscita**
- **la lunghezza massima (LM) di una riga nel file di uscita**

Il programma legga il file in ingresso e lo copi sul file di uscita centrando il testo di ogni singola riga rispetto alla lunghezza massima LM.

In pratica, le righe più lunghe di LM vanno troncate alla lunghezza LM e la rimanente parte deve essere considerata come una nuova riga.

Invece le righe più corte di LM devono essere centrate aggiungendo un'opportuna sequenza iniziale di spazi.

Esame del 11-lug-2000: soluzione

- **controllo gli argomenti**
- **apro i file**
- **leggo e controllo la lunghezza massima della riga (LM)**
- **leggo il file di input una riga per volta e per ogni riga:**
  - **se la sua lunghezza è < LM**
    - **stampo la riga centrata (ossia preceduta da un numero di spazi pari a (LM – LRIGA) / 2**
  - **altrimenti:**
    - **finché LRIGA > LM**
      - **ne stampo i primi LM caratteri**
      - **avanzo l'indice di inizio riga di LM caratteri**
    - **stampo centrata la restante parte**

e000711.c

Esame del 23-set-2000

Un file di testo contiene operazioni di acquisto e vendita di azioni. Ogni riga riporta una singola operazione, nel formato:

- **n. azioni acquistate o vendute (se n. negativo)**
- **valore di una azione in Euro (n. frazionario coi centesimi)**
- **sigla dell'azione**

Scrivere un programma che ha come primo argomento il nome del file che contiene le operazioni di compravendita e generi due file di testo (ACQUISTI.TXT e VENDITE.TXT) che riportino solo gli acquisti e le vendite, seguendo il formato originale.

In coda ad entrambi i file deve essere indicato il numero di operazioni riportate nel file ed il loro valore complessivo in Euro, con la precisione dei centesimi.

Un esempio di file input è il file COVE.TXT

Esame del 23-set-2000: soluzione

- **controllo dei parametri**
- **apertura del file di input e dei due file di output**
- **leggo tutte le righe del file di input**
- **per ogni riga:**
  - **leggo quantità e prezzo unitario (Q, P)**
  - **se (Q < 0)**
    - **copio la riga nel file delle vendite**
    - **aggiorno i contatori delle vendite**
  - **else**
    - **copio la riga nel file degli acquisti**
    - **aggiorno i contatori degli acquisti**
- **stampo i totali e chiudo i file**

e000923.c

Esercizio: fusione di due file

- **un primo file contiene su ogni riga il numero di matricola di uno studente**
- **un secondo file contiene su ogni riga il cognome di uno studente**
- **le righe dei due file si corrispondono**
- **i nomi dei file sono passati come parametri sulla linea di comando**
- **generare un unico file (terzo parametro) che contenga i dati di entrambi, ossia ogni sua riga contenga la matricola e poi il cognome di uno studente**

Esercizio: estensione della fusione

- **modificare il programma precedente in modo che:**
  - **legga e scriva cognome e nome**
  - **se il programma riceve solo due parametri allora scriva il risultato della fusione sull'unità di output standard**

Esercizio: diff

Fare un programma che:

- **confronti due file ed indichi se sono uguali o diversi**
- **i nomi dei file da confrontare sono passati come primo e secondo parametro sulla riga di comando**
- **se i file sono diversi, il programma indichi il numero della riga che contiene la prima differenza**
- **il programma restituisca 0 se i file sono uguali, 1 se sono diversi, 2 in caso di errore**

Esercizio: rig3

- **scrivere un programma che legga un file ed indichi se tutte le sue righe contengono esattamente 3 numeri interi**
- **le righe che non contengono tre numeri interi devono essere visualizzate precedute dal numero di riga**
- **il nome del file è passato come primo parametro sulla line di comando**

Esercizio: calcolo di un polinomio

- **data l'espressione $y = A x^2 + B x + C$**
- **sulla riga di comando sono passati i coefficienti A, B e C**
- **chiedere all'utente il valore X e calcolare il valore di Y corrispondente**
- **proseguire sino a quando l'utente introduce EOF**

Esercizio: conversione esadecimale - decimale

Viene fornito sulla riga di comando un numero esadecimale.

Scrivere un programma che fornisca in output il corrispondente numero decimale. Il programma deve segnalare se il numero fornito è scorretto.

Suggerimento:

$1F2_{16} = 1 \times 16^2 + F \times 16^1 + 2 \times 16^0$

$= 1 \times 16 \times 16 + F \times 16 + 2 \times 1$

$= (1 \times 16 + F) \times 16 + 2$

$= ((1) \times 16 + F) \times 16 + 2$

hexa.c

Esercizio

Un file (il cui nome è specificato come primo parametro sulla linea di comando) contiene una serie di dati reali, uno per riga.

Il file contiene al massimo 200 valori.

Presentare in output tutti i valori, uno per riga, preceduti da tre asterischi se il valore differisce di più del 50% dal valore medio, due asterischi se differisce di piu del 30%, un asterisco se differisce di più del 10%.

Nota: provare a farlo anche senza ipotizzare il numero massimo di valori presenti nel file.

Soluzione

- **controllo che ci siano i dati (argc)**
- **apro il file**
- **leggo i dati dal file e li metto in un vettore**
- **chiudo il file**
- **calcolo la media dei dati nel vettore**
- **per ogni elemento del vettore:**
  - **calcolo la sua differenza rispetto alla media**
  - **stampo gli asterischi necessari ed il suo valore**

diffmed.c

Alcune tipologie di lettura dei dati (I)

(caso 1) finché ci sono dati in input (EOF)

- **se devo leggere caratteri**

```
while (getchar != EOF) …
```

- **se devo leggere righe (eventualmente estraendone poi dei dati)**

```
while (gets != NULL) …
```

Alcune tipologie di lettura dei dati (II)

(caso 2) il numero di dati è noto (N):

- **uso un ciclo for**

```
for (i=0; i<N; i++) ...
```

- **il limite superiore N può essere:**
  - **fisso e noto a priori**

```
#define N 128
```

  - **passato tramite stdin**

```
gets(buf) + sscanf(buf,"%d",&N))
```

  - **letto da un file**

```
fopen + fgets(buf,...) + sscanf(buf,"%d",&N)
```

  - **passato come argomento**

```
sscanf (argv[1], "%d", &N))
```

Alcune tipologie di lettura dei dati (III)

(caso 3) fino ad un carattere (FINE) in prima colonna della riga:

- **leggo righe sino a trovarne una che inizia con FINE oppure sono arrivato alla fine dei dati (EOF)**
  - `while ((fgets(buf,…)!=NULL) && buf[0]!=FINE)`
- **FINE può essere**
  - **fisso e noto a priori**
    `#define FINE 'Z'`
  - **passato tramite stdin**
    `gets(buf) + FINE=buf[0]`
  - **letto da un file**
    `fopen + fgets(buf,...) + FINE=buf[0]`
  - **passato come (primo) argomento**
    `FINE=argv[1][0]`

Esercizio

Un file (nome passato come primo argomento) contiene:

- **nella prima riga il numero di giorni di cui sono forniti i dati**
- **i dati dei giorni in sequenza, ognuno con:**
  - **una prima riga che indica quanti dati sono forniti per quel giorn0**
  - **una riga per ogni dato fornito**

Calcolare la media di tutti i dati forniti.

Esempio di file:

```
2
   0
   1
       5
```

Input ASCII / numerico

L'input introdotto da tastiera o letto da un file di testo è interamente composto da codici ASCII, anche quando si tratta di dati numerici.

Usando argv[ ], gets( ), o il formato "%s" si leggono codici ASCII.

Usando il formato "%d" allora la funzione di lettura (es. sscanf()) effettua automaticamente una conversione alla base 10 … ma bisogna fare attenzione: se la stringa contiene caratteri non numerici non viene segnalato errore ma semplicemente la conversione si arresta alla parte numerica.

Esempio:

```
char buf[] = "123abc";
int n;

sscanf (buf, "%d", &n); /* n = 123 */
```

Esercizio

Cercare tra tutti i file di testo i cui nomi sono elencati sulla riga di comando quello che contiene la riga più lunga e presentare in output il nome di tale file, il numero della riga più lunga e la sua lunghezza.

Per semplificare l'esercizio, è possibile fare un'ipotesi sulla lunghezza di una singola riga (es. non sia più lunga di 255 caratteri), ma in questo caso il voto massimo sarà 27.

Soluzione

- **esaminando un file per volta, devo tenere traccia del nome del file, numero e lunghezza della riga più lunga trovata sino ad un certo punto:**

```
struct {
  int n_file; // posizione in argv[]
  int n_riga;
  int l_riga;
} max;
```

- **controllo dei dati (c'è almeno un file?)**
- **per ogni file:**
  - **cerco la riga più lunga ed eventualmente aggiorno max**
- **stampo il risultato**

lrigaf.c

Esercizio (uniqn)

Scrivere un programma che:

- **lavori su un file di input ed uno di output, specificati rispettivamente come penultimo ed ultimo parametro sulla linea di comando**
- **esamini il file di input e copi sul file di output solo le righe che non sono "duplicate"**
- **si intendono "duplicate" righe consecutive che sono completamente uguali, oppure sono uguali nei primi N caratteri (se c'è l'argomento -iN) oppure negli ultimi N caratteri (se c'è l'argomento -fN)**

Problema (tra apici lettere maiuscole)

Copiare l'input sull'output convertendo in maiuscolo le parole racchiuse tra doppi apici.

Sottoproblema:

- **come ricordare che in precedenza si è trovato il carattere "doppio apice"?**

Pattern – il concetto di stato

- **tutte le volte che**
  - **si svolge un'azione su una sequenza di input e**
  - **l'azione deve cambiare a seconda dei dati incontrati**
- **allora è utile:**
  - **introdurre una variabile che rappresenti lo "stato" dell'elaborazione (meglio se di tipo enum)**
  - **svolgere i diversi tipi di elaborazione in modo condizionato (con una cascata di if oppure con un'istruzione switch)**
- **per evitare di sbagliare nell'elaborazione, è meglio dare nomi significativi ai vari stati**

Il concetto di stato - scheletro

```
enum {S0, S1, S2} stato;

stato = S0; /* lo stato iniziale */
. . .
if (stato == S0)
{
  /* azioni da svolgere nello stato S0 */
}
else if (stato == S1)
{
  /* azioni da svolgere nello stato S1 */
}
else if (stato == S2)
  . . .
else
  /* bug! stato impossibile */
```

Il concetto di stato - azioni

- **le azioni svolte in ciascun stato sono di due tipi:**
  - **azioni di calcolo, relative alle operazioni da svolgere in un certo stato**
  - **azioni di cambiamento di stato (ossia del tipo stato = nuovo_stato) che permettono di cambiare la sequenza delle operazioni, transitando da uno stato all'altro**

Diagramma di stato

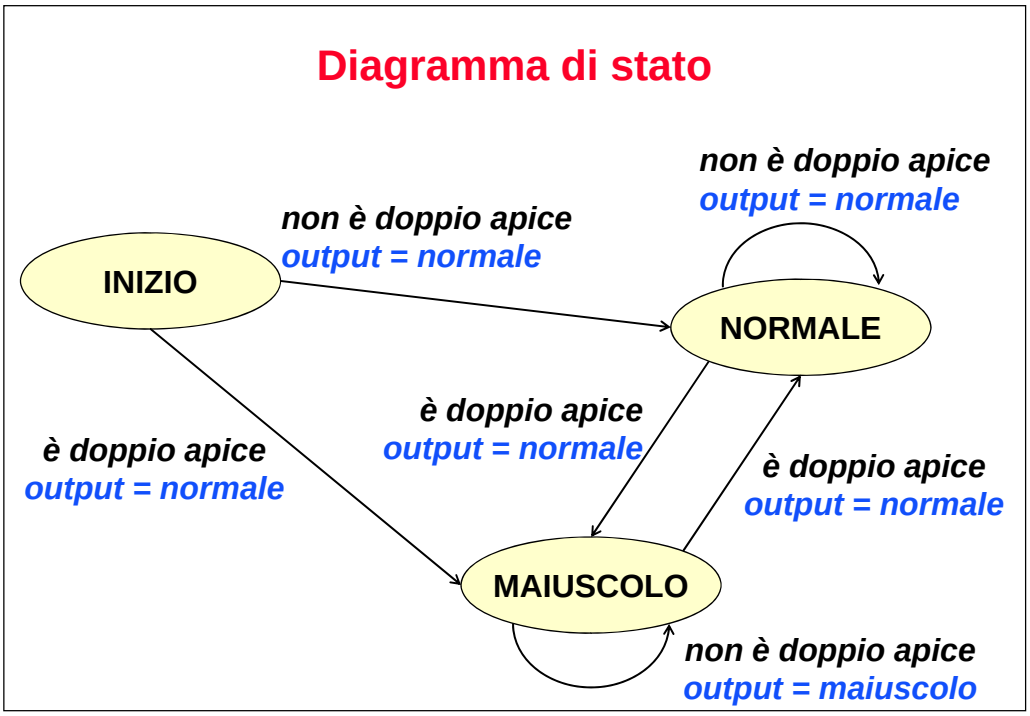
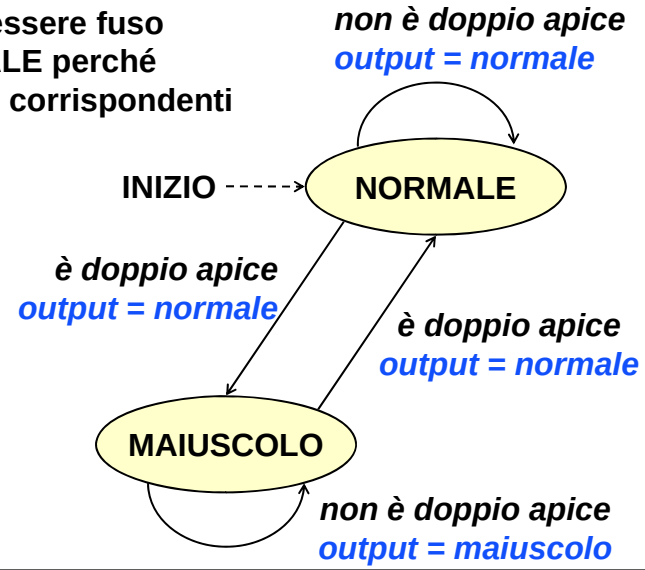


Diagramma di stato (ridotto)

lo stato INIZIO può essere fuso con lo stato NORMALE perché hanno archi e azioni corrispondenti



Tra apici lettere maiuscole: soluzione

```
enum {Normale, Maiuscolo} stato;

stato = Normale;
while ( ( xc=getchar() ) != EOF)
if (stato == Normale)
{
  /* copia xc in output inalterato */
  /* se xc == '"' allora stato=Maiuscolo */
}
else if (stato == Maiuscolo)
{
  /* copia xc in output maiuscolo */
  /* se xc == '"' allora stato=Normale */
}
else
  /* bug! stato impossibile – bye! */
```

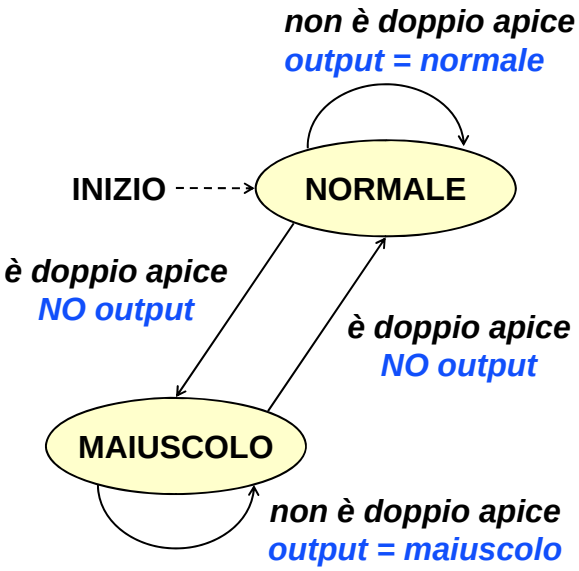
maiuapic.c

Esercizio

Copiare l'input sull'output convertendo in maiuscolo le parole racchiuse tra doppi apici senza copiare i doppi apici.

maiuapic2.c

Diagramma di stato (maiuapic2.c)



maiuapic2.c

Esercizio - commenti (assembler)

Premessa: in alcuni linguaggi di programmazione (ad esempio in Assembler) è possibile introdurre commenti che iniziano con un carattere particolare e terminano a fine riga.

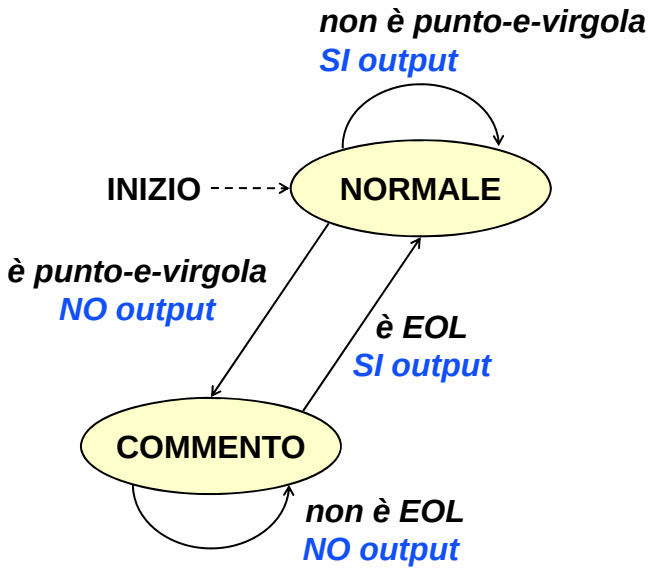
Esercizio

Sapendo che in Assembler i commenti iniziano col carattere "punto e virgola" e terminano a fine riga, scrivere un programma che copi l'input sull'output eliminando tutti i commenti in stile Assembler.

Si noti che il "punto e virgola" è anch'esso parte del commento e deve quindi essere omesso dall'output.

commass.c

Diagramma di stato (commass.c)



Esercizio - commenti (C++)

Premessa: in alcuni linguaggi di programmazione (ad esempio in C++) è possibile introdurre commenti che iniziano con una sequenza di caratteri particolare e proseguono solo fino a fine riga.

Esercizio

Sapendo che in C++ i commenti iniziano con la sequenza di caratteri "//" e terminano a fine riga, scrivere un programma che copi l'input sull'output eliminando tutti i commenti in stile C++.

Si noti che la sequenza "//" è anch'essa parte del commento e deve quindi essere omessa dall'output.

commcpp.c

Esercizio - commenti (C)

Sapendo che in C i commenti iniziano con la sequenza di caratteri "/*" e terminano con la sequenza di caratteri "*/" , scrivere un programma che copi l'input sull'output eliminando tutti i commenti in stile C.

Si noti che le sequenze "/*" e "*/" sono anch'esse parte del commento e devono quindi essere omesse dall'output.

Esercizio – spaziatura unica

Scrivere un programma che:

- **copi tutto l'input sull'output**
- **se in input sono presenti sequenze di due o più spazi, in output devono essere sostituite da uno spazio solo**
- **se una riga inizia o termina con degli spazi questi devono essere tutti eliminati nella copia**

unospaz.c

Tema d'esame del 21/2/2011: traccia

- **leggo parametri da riga di comando [myLat, myLong]**
- **apro file "montagne.txt" in lettura [afh]**
- **while (fgets(afh,buf) != NULL) {**
  - **carico in una struct i dati della montagna corrente [monte]**
  - **calcolo la distanza (myLat, myLong, monte) [dist]**
  - **if (primo_monte)**
    - **mindist = dist, primo_monte = falso;**
  - **else**
    - **if (dist < mindist) minMonte = monte;**
- **}**
- **stampo dati minMonte**
- **chiudo file afh**

e110221_noc.c
e110221.c