

Primi programmi in C

Laboratorio di algoritmi e strutture dati
Docente: Violetta Lonati

Nota: ricordate le opzioni principali del comando `gcc` (per eventuali dubbi, consultate il manuale on-line `man gcc`):

- `-o` per salvare l'*output* in un file specificato,
- `-c` per arrestare la compilazione prima della fase di *link*,
- `-s` per arrestare la compilazione prima della fase di *assemble*,
- `-E` per arrestare la compilazione dopo la fase di *preprocessing*,
- `-w`, con parametro `all`, per la segnalazione di avvertimenti (warning),
- `-l`, con parametro `m`, per linkare le librerie matematiche.

Utilizzate l'editor `gedit` per creare i file sorgenti corrispondenti agli esercizi seguenti.

1 Esercizi introduttivi

1.1 Conversione temperatura

Scrivete un programma che legga una temperatura in gradi Farenheit e stampi l'equivalente in gradi centigradi.

1.2 Equazione di secondo grado

Scrivete un programma che legga tre coefficienti a, b, c e calcoli le soluzioni (complesse) dell'equazione $ax^2 + bx + c$. Può essere utile includere il file di intestazione `math.h` della libreria standard, contenente la funzione `sqrt` per il calcolo della radice quadrata.

1.3 Alfabeto italiano

Scrivete un programma che legga un carattere e stabilisca se si tratta di una vocale dell'alfabeto italiano, di una consonante dell'alfabeto italiano, o un carattere che non fa parte dell'alfabeto italiano. Per la lettura di un carattere, potete usare l'istruzione `scanf("%c", &c)`; dove `c` è una variabile di tipo `char`. Può essere utile inoltre includere il file di intestazione `ctype.h` della libreria standard, contenente alcune funzioni per l'analisi e l'elaborazione di caratteri, quali ad esempio `tolower`.

1.4 Classificazione triangoli

Scrivete un programma che, dati tre numeri, stabilisce se questi possono definire le lunghezze dei lati di un triangolo e, in caso affermativo, classifica il triangolo come equilatero, isoscele o scaleno. Ricordate questa proprietà fondamentale: in un triangolo, ogni lato è più corto della somma degli altri due.

1.5 Somma di numeri

Scrivete un programma che legga una serie di numeri positivi terminata da 0 e ne calcoli la somma.

1.6 Media di numeri

Modificate il programma precedente in modo da calcolare la media dei numeri inseriti.

1.7 Somma di esattamente 10 numeri non nulli

Scrivete un programma che legga una serie di numeri e ne calcoli la somma; il programma deve terminare dopo aver letto esattamente 10 numeri diversi da 0. Può essere utile l'utilizzo delle istruzioni `continue` e/o `break`.

1.8 Cerchio

Scrivete un programma che legga (in una variabile di tipo `float`) il raggio di un cerchio e ne stampi l'area.

Tenete presente che il file `math.h` della libreria standard contiene la definizione della costante `M_PI`, pari al valore di π (il rapporto tra la circonferenza ed il diametro).

1.9 Quadrati perfetti

Scrivete un programma che stampi la sequenza crescente dei primi 10 quadrati perfetti (ossia numeri x tali che $x = y^2$ per qualche numero naturale y). Usate una macro di valore 10 per rendere il programma più facile da modificare!

1.10 Massimo tra due numeri

Scrivete un programma che legga due numeri interi e stabilisca qual è il massimo, usando l'operatore condizionale `expr1 ? expr2 : expr3`.

1.11 Ordine alfabetico

Scrivete un programma che legga due lettere maiuscole e stampi la loro distanza nell'ordine alfabetico. Ad esempio, su ingresso **A C**, il programma deve stampare 3, su ingresso **F B**, il programma deve stampare 5. Ricordate che i caratteri (`char`) in C sono rappresentati come (piccoli) interi e osservate che, secondo la codifica ASCII, non c'è soluzione di continuità e non ci sono altri caratteri tra la A e la Z (cosa di cui potete convincervi con il comando `man ascii`).

2 Esercizi da svolgere in laboratorio

2.1 Operatore `sizeof` e `limits.h`

L'operatore unario `sizeof` applicato ad una variabile ha per valore la dimensione della variabile a cui è applicato espresso in byte. Così, ad esempio (sull'architettura delle macchine presenti in laboratorio), se la variabile `x` è di tipo `int`, allora l'espressione `sizeof(x)` ha valore 4.

Il file di intestazione `limits.h` contiene (tra le altre cose) le definizioni (ottenute grazie alla direttiva `#define`) dei valori limite, per l'architettura corrente, dei tipi interi e carattere. Ad esempio, il seguente programma stampa l'intervallo di valori possibili per le variabili di tipo intero con segno.

```

#include <stdio.h>
#include <limits.h>

int main( void ) {
    printf( "%d..%d\n", INT_MIN, INT_MAX );

    return 0;
}

```

Le definizioni dei limiti per alcuni degli altri tipi fondamentali si chiamano: `SCHAR_MIN`, `SCHAR_MAX` e `UCHAR_MAX` per il tipo carattere con e senza segno, `SHRT_MIN`, `SHRT_MAX` e `USHRT_MAX` per il tipo intero breve, con e senza segno.

Scrivete un programma che dichiari una variabile per ciascuno dei tipi fondamentali e delle sue rispettive varianti `long` e `short` (qualora ammissibili), e ne stampi la dimensione in byte ottenuta tramite l'operatore `sizeof` e l'intervallo di valori possibili per tali tipi.

2.2 Indovina il numero

Il gioco *Indovina il numero* funziona come segue: un giocatore *A* pensa a un numero intero x (con $0 \leq x \leq 1000$), e l'altro giocatore, *B*, lo deve indovinare. Per farlo, *B* pone domande del tipo “Il numero è y ?", cui *A* può rispondere `=` (per indicare che il numero è stato indovinato), oppure `<` (per indicare che x è minore del numero y), oppure `>` (per indicare che x è maggiore del numero y).

Scrivete un programma che giochi come giocatore *B*, seguendo questa strategia: gli estremi entro cui può stare il valore da indovinare vengono memorizzati in due variabili che verranno aggiornate ad ogni risposta dell'utente; ogni volta, il programma chiede se il valore è quello in mezzo all'intervallo e, a seconda della risposta dell'utente, il programma esce oppure modifica i valori di un estremo in modo da restringere l'intervallo.

Esempio di funzionamento:

```

Il numero è 500? <
Il numero è 249? <
Il numero è 124? <
Il numero è 61? <
Il numero è 30? >
Il numero è 45? <
Il numero è 37? =

```

NOTA: Questo algoritmo di ricerca vale per ogni sequenza ordinata ed è chiamato algoritmo di *ricerca dicotomica*.

2.3 Il cifrario di Cesare

Svetonio nella *Vita dei dodici Cesari* racconta che Giulio Cesare usava per le sue corrispondenze riservate un codice di sostituzione molto semplice, nel quale la lettera chiara veniva sostituita dalla lettera che la segue di tre posti nell'alfabeto: la lettera A è sostituita dalla D, la B dalla E e così via fino alle ultime lettere che sono cifrate con le prime.

Più in generale si dice codice di Cesare un codice nel quale la lettera del messaggio chiaro viene spostata di un numero fisso k di posti, non necessariamente tre.

Dovete scrivere un programma che legga (usando `scanf`) in input il numero k (chiave di cifratura) e il testo da cifrare, sotto forma di una sequenza di caratteri terminata da `.` (mediante la funzione `getchar()` inclusa

in `stdio.h`), e che emetta in output il testo cifrato; il programma deve cifrare solo le lettere dell'alfabeto, mantenendo minuscole le minuscole, e maiuscole le maiuscole, mentre deve lasciare inalterati gli altri simboli.

Per provare se il programma funziona, cifrate un messaggio con una certa chiave k e poi applicate al risultato una nuova cifratura con chiave $26 - k$: il risultato dovrebbe essere la stringa originale.

Esempio di funzionamento

```
5 mamma li Turchi
```

```
rfrrf qn Yzwhmn
```

```
21 rfrrf qn Yzwhmn
```

```
mamma li Turchi
```

2.4 Disegni

Questo esercizio richiede di scrivere dei programmi che stampino delle figure, secondo gli schemi sotto definiti. L'utente dovrà inserire un intero n che definisce la dimensione della figura da disegnare.

1. Righe alternate: la figura è ottenuta alternando righe costituite solo da + e righe costituite solo da o:

$n = 3$
+ + +
o o o
+ + +

$n = 5$
+ + + + +
o o o o o
+ + + + +
o o o o o
+ + + + +

2. Caratteri alternati: la figura è ottenuta alternando caratteri o e +:

$n = 5$
o + o + o
+ o + o +
o + o + o
+ o + o +
o + o + o

$n = 6$
o + o + o +
+ o + o + o
o + o + o +
+ o + o + o
o + o + o +
+ o + o + o

3. Triangolo: il triangolo sotto la diagonale con direzione alto/sx verso basso/dx è formato da o, il triangolo sopra la diagonale da +, la diagonale da |.

$n = 5$
+ + + +
o + + +
o o + +
o o o +
o o o o

$n = 6$
+ + + + +
o + + + +
o o + + +
o o o + +
o o o o +
o o o o o

4. Rombo: la figura rappresenta un rombo avente i lati nei punti medi dei lati del quadrato. Notate che queste figure sono definite solo per n dispari!

$n = 5$
o
o o o
o o o o o
o o o
o

$n = 7$
o
o o o
o o o o o
o o o o o o o
o o o o o
o o o
o

3 Altri esercizi

3.1 Divisori e numeri primi

1. Scrivete un programma che stampi la sequenza decrescente dei numeri divisori di n , dove n è un numero inserito in input. Modificate il programma in modo che stampi anche il numero di divisori di n .
2. Scrivete un altro programma che, usando un ciclo `for`, stabilisca se un numero n è primo (ovvero ha per divisori solo 1 e se stesso) oppure no. Cercate di ridurre il numero di istruzioni da eseguire! Scrivete una nuova versione del programma precedente usando un ciclo `while`.
3. Scrivete un programma che scomponga in fattori primi un numero dato in input.

3.2 Calendario

Scrivete un programma che stampi un calendario mensile. L'utente deve specificare il nome del mese e il giorno della settimana in cui comincia il mese. Per semplicità considerate solo anni non bisestili...

Esempio di funzionamento:

```
Inserisci il numero del mese (1 = gennaio, ..., 12 = dicembre): 2
Inserisci il giorno di inizio mese (1 = lunedì, ..., 7 = domenica): 4
L   M   M   G   V   S   D
                          1   2   3   4
 5   6   7   8   9   10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28
```

3.3 Espressioni ben parentesizzate

Una successione di caratteri è un'*espressione ben parentesizzata* se, per ogni prefisso della successione stessa (cioè, per ogni possibile segmento iniziale della successione), il numero di parentesi aperte "(" è maggiore o uguale al numero di parentesi chiuse ")", e se, complessivamente, il numero di parentesi aperte è uguale al numero di parentesi chiuse.

Scrivete un programma che legga (mediante la funzione `getchar()` inclusa in `stdio.h`) una sequenza di caratteri terminata da . e determini se essa è un'espressione ben parentesizzata. In caso negativo, il programma dovrà stampare in quale posizione ha identificato un errore, e il tipo di errore.

Esempio di funzionamento:

Stringa: ((1)abb(3(2a)4(b))5).

La stringa è un'espressione ben parentesizzata

Stringa: ((1)abb(3(2a)))4(b5).

La stringa non è un'espressione ben parentesizzata:

Carattere 19: mancano parentesi chiuse alla fine!

Stringa: ((1)abb(3))(2)a4(b)5).

La stringa non è un'espressione ben parentesizzata:

Carattere 15: troppe parentesi chiuse!

3.4 Intervallo di tempo

Scrivete un programma che calcoli l'intervallo di tempo compreso tra due orari. Assumete che sia gli orari che l'intervallo di tempo siano rappresentati nel formato a 24 ore hh:mm:ss. E' possibile usare short int invece che int?

3.5 Conversione orario

Scrivete un programma che, dato un orario in formato a 24 ore hh:mm, fornisca il corrispondente orario nel formato AM/PM e viceversa. Ricordate che secondo il formato AM/PM, le 24 ore del giorno sono suddivise in due periodi chiamati AM (ante meridiem) e PM (post meridiem): ogni periodo consiste di 12 ore numerate con 12 (usato come 0), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. L'orario 12:00 AM indica la mezzanotte, l'orario 12:00 PM indica il mezzogiorno.

