Prototipi delle Funzioni

In C è possibile definire un prototipo di funzione prima di fornire l'implementazione del corpo della funzione.
Un prototipo di funzione consiste nella definizione del tipo di ritorno della funzione, l'assegnazione di un nome (identificatore valido) alla funzione, e l'elenco dei parametri che la funzione deve ricevere al momento della chiamata, con il relativo tipo.

```
int massimo_di_tre (int, int, int);
int minimo_di_tre (int, int, int);
```

I nomi dei parametri sono opzionali (possono essere inseriti ma vengono ignorati dal compilatore).

Definizione di una Funzione

Per utilizzare una funzione all'interno di un programma la sola definizione del prototipo non è ovviamente sufficiente. La funzione deve essere definita fornendo l'implementazione del suo corpo nella forma:

```
tipo-di-ritorno nome-funzione (lista-parametri) {
  dichiarazioni
  istruzioni
}
```

Il tipo di ritorno può essere omesso se la funzione non restituisce nulla (meglio usare il tipo `void` in questo caso).

```
int massimo_di_tre (int a, int b, int c){
  int massimo = a;
  if (massimo < b) massimo = b;
  return massimo < c ? c : massimo;
}
```

Passaggio dei parametri per Valore

I parametri delle funzioni in C vengono passati per VALORE. Ogni parametro della funzione diventa una variabile, visibile all'interno della funzione stessa. Al momento della chiamata di una funzione, il valore attuale del parametro fornito in input viene copiato nella variabile corrispondente.

```
#include<stdio.h>

int b=1;
int effetti_collaterali (int);
int main (void){
int a=5;
printf("a vale %d e b vale %d\n", a, b);
printf("effetti_collaterali(a) restituisce %d\n",
                              effetti_collaterali(a));
printf("a vale %d e b vale %d\n", a, b);
}
int effetti_collaterali(int a){return ++a + b++;}
```

Passaggio dei parametri per Valore

```
#include<stdio.h>

int b=1;
int effetti_collaterali (int);
int main (void){
  int a=5;
  printf("a vale %d e b vale %d\n", a, b);
  printf("effetti_collaterali(a) restituisce %d\n",
                              effetti_collaterali(a));
  printf("a vale %d e b vale %d\n", a, b);
  return 0;
}
int effetti_collaterali(int a){return ++a + b++;}
```

Passaggio dei parametri per Valore

```
#include<stdio.h>

int b=1;
int effetti_collaterali (int);
int main (void){
  int a=5;
  printf("a vale %d e b vale %d\n", a, b);
  printf("effetti_collaterali(a) restituisce %d\n",
                              effetti_collaterali(a));
  printf("a vale %d e b vale %d\n", a, b);
  return 0;
}
int effetti_collaterali(int a){return ++a + b++;}
```

```
 a vale 5 e b vale 1
 effetti_collaterali(a) restituisce 7
 a vale 5 e b vale 2
```

Cenni sulla visibilità delle variabili

Nella funzione `effetti_collaterali` abbiamo utilizzato una variabile *globale*. Le variabili definite all'esterno di qualunque funzione sono visibili a tutte le funzioni, quelle definite all'interno di una funzione sono visibili solamente all'interno della funzione stessa.

Cenni sulla visibilità delle variabili

```
#include <stdio.h>
int b=1;
int effetti_collaterali (int);
int main (void){
  int a=5;
  printf("a vale %d e b vale %d\n", a, b);
  printf("effetti_collaterali(a) restituisce %d\n",
                              effetti_collaterali(a));
  printf("a vale %d e b vale %d\n", a, b);
  return 0;
}
int effetti_collaterali(int b){return ++a + b++;}
```

Cenni sulla visibilità delle variabili

```
#include <stdio.h>
int b=1;
int effetti_collaterali (int);
int main (void){
  int a=5;
  printf("a vale %d e b vale %d\n", a, b);
  printf("effetti_collaterali(a) restituisce %d\n",
                              effetti_collaterali(a));
  printf("a vale %d e b vale %d\n", a, b);
  return 0;
}
int effetti_collaterali(int b){return ++a + b++;}
```

```
effetti_collaterali_err.c: In function 'effetti_collaterali':
effetti_collaterali_err.c:15: error: 'a' undeclared (first use in
 this function)
effetti_collaterali_err.c:15: error: (Each undeclared identifier is
 reported only once
effetti_collaterali_err.c:15: error: for each function it appears in.)
```

Cenni sulla visibilità delle variabili

```
#include <stdio.h>
int b=1;

int main (void){
  int a=5;
  int effetti_collaterali(int b){return ++a + b++;}

  printf("a vale %d e b vale %d\n", a, b);
  printf("effetti_collaterali(a) restituisce %d\n",
                          effetti_collaterali(a));
  printf("a vale %d e b vale %d\n", a, b);
  return 0;
}
```

Cenni sulla visibilità delle variabili

```
#include <stdio.h>
int b=1;

int main (void){
  int a=5;
  int effetti_collaterali(int b){return ++a + b++;}

  printf("a vale %d e b vale %d\n", a, b);
  printf("effetti_collaterali(a) restituisce %d\n",
                          effetti_collaterali(a));
  printf("a vale %d e b vale %d\n", a, b);
  return 0;
}
```

```
a vale 5 e b vale 1
effetti_collaterali(a) restituisce 11
a vale 6 e b vale 1
```

Le variabili `static`

La parola chiave `static` prima della dichiarazione di una variabile rende tale variabile permanente: la locazione in memoria di tale variabile viene allocata all'inizio dell'esecuzione del programma e il suo valore viene mantenuto durante tutta l'esecuzione.

```
int massimo_visto (int i){
  static int massimo = 0;
  if (i>massimo) massimo=i;
  return massimo;
}
```

Esempio

```
#include <stdio.h>

int massimo_visto(int);

int main(void) {
  int val;
  while(1){
    printf("Inserire un intero positivo (-1 per terminare)\n");
    scanf("%d", &val);
    if (-1 == val) break;
    if (0 < val) printf("Il massimo inserito finora e'
                                    %d\n",massimo_visto(val));
  }
  return 0;
}
int massimo_visto (int i){
  static int massimo = 0;
  if (i>massimo) massimo=i;
  return massimo;
}
```

Esempio

```
Inserire un intero positivo (-1 per terminare)
10
Il massimo inserito finora e' 10
Inserire un intero positivo (-1 per terminare)
5
Il massimo inserito finora e' 10
Inserire un intero positivo (-1 per terminare)
12
Il massimo inserito finora e' 12
Inserire un intero positivo (-1 per terminare)
-1
```

Permanenza e visibilità

La variabili globali sono permanenti (come le variabili dichiarate `static`). Le varibili definite all'interno di una funzione sono invece allocate automaticamente (`auto` o `register`), e pertanto il loro ultimo valore viene perso quando si esce dalla funzione nella quale sono state definite.
La permanenza di una varibile non cambia la sua visibilità.

Esercizio

Scrivere un programma in linguaggio C che legga da tastiera 2 numeri interi positivi (a e b) e disegni un rettangolo di a colonne e b righe usando il carattere "*".

Esercizio

Scrivere un programma in linguaggio C che legga da tastiera una sequenza di lunghezza ignota a priori di numeri interi positivi. Il programma, a partire dal primo numero introdotto, stampa ogni volta la media di tutti i numeri introdotti. Terminare quando il numero inserito è negativo.
Per implementare questo esercizio potete definire una sola variabile nella funzione main, e non potete utilizzare variabili globali.

Esercizio

Scrivere un programma in linguaggio C richieda un numero n in input e stampi la tavola pitagorica (le tabelline) per i numeri da 1 ad n (incolonnati), allineati a destra su ogni colonna.
Potete usare un numero fisso di posizioni per scrivere il numero, o meglio, calcolare quante ne servono prima di cominciare a stampare (in questo caso non si può usare il "%10u").

Esercizio

Scrivere un programma in linguaggio C che legga da tastiera un valore intero positivo l e stampi un quadrato di lato l usando il carattere "*" per il bordo e il carattere "o" internamente. Se l è maggiore di 5, il programma stampa anche un altro quadrato di lato $l - 4$ centrato su quello di lato l nella stessa maniera.
Usate una funzione `carattere_per` che dato l e la posizione (riga e colonna) restituisca il carattere adeguato. Per $l = 7$ deve stampare:

```
*******
*ooooo*
*o***o*
*o*o*o*
*o***o*
*ooooo*
*******
```

Come possiamo modificare il programma in modo da non dover passare il valore l alla funzione?

Esercizio

Modificare la funzione `carattere_per` in modo che stampi un rombo all'interno del quadrato di lato *l* se *l* è dispari e maggiore o uguale a 5 (per *l* pari la stampa rimane quella dell'esercizio precedente). Esempio:

```
*******
*oo*oo*
*o*o*o*
**ooo**
*o*o*o*
*oo*oo*
*******
```