Le funzioni in C

I programmi C sono costituiti da **definizioni di variabili** e **funzioni**.

Una definizione di funzione ha il seguente formato:

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

Le definizioni di funzioni possono comparire in qualsiasi ordine all'interno di uno o più **file sorgente**, ma non possono essere spezzate in più file.

Esempio

Il seguente programma è costituito da una funzione `power(m,n)` che eleva un intero m alla potenza intera n, e dalla funzione speciale `main` che utilizza `power`.

```
# include <stdio.h>
int power(int m, int n); /* prototipo */
main()
{
   int i;
   for  (i=0; i<10; ++i)
      printf("%d %d %d\n", i, power(2,i), power(-3,i));
   return 0;
}
int power(int base, int n)
{
   int i,p;
   p=1;
   for (i=1; i<=n; ++i)
      p=p*base;
   return p; \* restituisce il valore di p al chiamante *\
}
```

Analisi dell'esempio

La dichiarazione `int power(int m, int n)` all'inizio del programma è detta **prototipo** della funzione, indica che `power` si aspetta due argomenti interi e restituisce un intero. Il prototipo deve essere in accordo con la definizione della funzione stessa (a parte per i nomi dei parametri).

La prima linea della funzione `power` dichiara i nomi e i tipi dei parametri e il tipo del risultato. I nomi dei parametri sono locali a `power`, così come le variabili i, p dichiarate all'interno di `power`.

L'istruzione `return` ritorna il controllo al chiamante, eventualmente restuituendo il valore specificato.
Per le funzioni in cui l'istruzione `return` non compare, il controllo ritorna al chiamante alla fine dell'esecuzione della funzione.
Anche la funzione speciale `main` può avere un'istruzione `return`, che ritorna il controllo al chiamante, cioè l'ambiente in cui il programma è eseguito.

Argomenti: chiamata per valore e per riferimento

In C tutti gli argomenti delle funzioni, che **non** sono **vettori** nè **puntatori**, sono passati **per valore**. Cioè le funzioni lavorano su copie dei parametri e non modificano i parametri passati dal chiamante.

Al contrario, argomenti **vettore** o **puntatore** sono passati **per riferimento**, cioè viene passato l'**indirizzo** dei parametri e la funzione lavora sui parametri originali.

Variabili e scope

Le variabili dichiarate all'**interno** delle funzioni sono **locali** alle funzioni e sono dette **automatiche**, in quanto sono create al momento della chiamata della funzione e cessano di esistere quando questa termina.
Perciò una variabile automatica deve essere sempre inizializzata ad ogni chiamata, altrimenti può provocare errore.

Le variabili dichiarate come `static` sono variabili che conservano il loro valore tra una chiamata e l'altra di una funzione.

Le variabili **esterne** (globali) sono **definite al di fuori** delle funzioni. Tali variabili devono essere anche **dichiarate all'interno** di ogni funzione che le usa come variabili `extern`.

Le variabili dichiarate come `register` sono collocate in **registri** della macchina e quindi permettono un accesso più rapido. Ci sono però delle limitazioni (e.g. non si può accedere all'indirizzo di una variabile register); inoltre il compilatore può ignorare la dichiarazione `register`.

Esempio

Programma che legge un insieme di linee di testo e stampa la più lunga.

```
#include <stdio.h>
#define MAXLINE 1000  /* lunghezza massima di una linea */

int getline(char line[], int maxline);
void copy(char to[], char from[]);

main()
{
   int len;  /* lunghezza della linea corrente */
   int max;  /* massima lunghezza trovata finora */
   char line[MAXLINE];  /* linea di input corrente */
   char longest[MAXLINE];  /* linea piu' lunga corrente */

   max=0;
   while ((len=getline(line, MAXLINE)) > 0)
      if (len > max)
      {  max=len;
         copy(longest, line);
      }
```

. . . esempio

```
    if (max > 0)  /* c'era almeno una linea in input */
        printf("%s", longest);
    return 0;
}
```

Esercizio: definire le funzioni

```
/* getline: legge e carica in s una linea, ritorna la lunghezza */
int getline(char s[], int lim)

/* copy: copia from in to; assume che to sia sufficientemente ampio */
void copy(char to[], char from[])
```

Versione alternativa

Versione alternativa del programma per la stampa della linea più lunga, che definisce le variabili `line`, `longest` e `max` come variabili esterne.

```
#include <stdio.h>
#define MAXLINE 1000  /* lunghezza massima di una linea */

int max;  /* massima lunghezza trovata finore */
char line[MAXLINE];  /* linea di input corrente */
char longest[MAXLINE];  /* linea piu‘ lunga corrente */

int getline(void);
void copy(void);

main()
{
   int len;
   extern int max;
   extern char longest[MAXLINE];

   max=0;
```

...

```
   while ((len=getline()) > 0)
      if (len > max)
      {  max=len;
         copy();
      }
   if (max > 0)  /* c'era almeno una linea in input */
      printf("%s", longest);
   return 0;
}
```

Esercizio: definire le funzioni

```
int getline(void)

void copy(void)
```

Esercizi

Scrivete le seguenti funzioni:

1. `reverse(s)`, che inverte la stringa di caratteri s. Usate tale funzione per scrivere un programma che inverte le linee di un testo in input.

2. `strindex(s,t)` che restituisce la posizione, cioè l'indice di inizio della stringa t nel vettore s, oppure -1 se t non compare in s.

3. `strrindex(s,t)` che restituisce la posizione dell'occorrenza più a destra di t in s, oppure -1 se t non compare in s.