Le funzioni

Prof. Orazio Mirabella

Sommario

- Definizione e dichiarazione
- Parametri attuali e formali
- Visibilita'
- Return e void
- Rappresentazione in memoria
- Passaggio per valore e per riferimento
- Passaggio di array a funzioni
- Passaggio di strutture a funzioni

Le funzioni

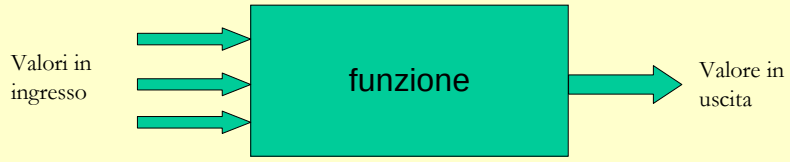
- Per *funzione* s'intende un segmento di programma autocontenuto che esegue un lavoro specifico ben definito.
- L'uso delle funzioni permette di spezzare un grosso programma in un insieme di componenti più piccole, ognuna con un compito ben definito.
- Le funzioni si utilizzano anche per evitare di replicare porzioni di codice sorgente: chiamare una funzione significa mandare in esecuzione la porzione di codice corrispondente.

Le funzioni

- E' possibile poi creare delle librerie, cioè delle raccolte di funzioni che possono essere utilizzate senza essere a conoscenza dei dettagli implementativi.
- Es: le funzioni *scanf()* e *printf()*, la cui dichiarazione è contenuta nel file stdio.h

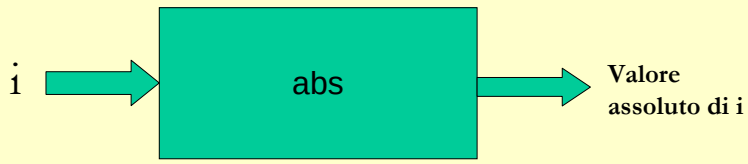
Le funzioni

- Una funzione può essere pensata come una scatola nera, che a determinati valori in ingresso fa corrispondere un determinato valore in uscita.



Esempio

- La funzione *abs(i)*. Considerandola come una scatola nera, tutto ciò che dobbiamo sapere è che inserendo come argomento *i* di tale funzione un numero intero essa ne ritorna il valore assoluto.



Dichiarazione di una funzione

- Una funzione viene dichiarata nel seguente modo:

Tipo_ritorno nome_funz (tipo_par1, . . . , tipo_parN);
Oppure:
Tipo_ritorno nome_funz (tipo_par1 par1, . . . , tipo_parN parN);

La dichiarazione introduce il nome della funzione,che in questo modo può essere utilizzato dal programma, ma presuppone che da qualche altra parte ne esista la definizione, altrimenti quel nome resterebbe privo di significato ed il compilatore segnalerebbe un errore.

Il nome del parametro formale, però, è assolutamente superfluo. Se in una dichiarazione di una funzione si specificano anche i nomi dei parametri i il compilatore semplicemente li ignora.

Definizione di una funzione 1/2

- In generale una funzione viene definita nel seguente modo:

```
Tipo_ritorno nome_funz (tipo_par1 par1, . . . ,
  tipo_parN parN)
{
  // corpo della funzione
}
```

La definizione stabilisce il nome della funzione, i valori in ingresso su cui agisce – detti *parametri formali* –, il blocco di istruzioni che ne costituiscono il contenuto, e l'eventuale valore di ritorno

Definizione di una funzione 2/2

➔Per ogni dichiarazione introdotta nel programma occorre una definizione, ma ricordiamo che in C NON è ammesso che più funzioni abbiano lo stesso nome. Per esempio, le due definizioni, poste all'interno dello stesso programma:

```
Double cubo(float c) {
  return(c*c*c)
}
```

```
Double cubo(int c) {
      return (c*c*c)
}
```

darebbero luogo ad un errore!!!

Un esempio

```
#include <stdio.h>
double cubo(float); /*dichiarazione*/
main()
{
float a;
double b;
printf("Inserisci un numero: ");
scanf("%f", &a);
b = cubo(a); /* invocazione della funzione*/
printf("%f elevato al cubo è uguale a %f", a, b);
}
double cubo(float c) /*definizione della funzione*/
{
return (c*c*c); /*ritorno di un valore*/
}
```

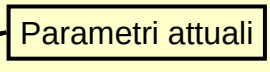
Parametri Formali e Parametri Attuali

- Parametri formali
  - Sono specificati nella dichiarazione della funzione
- Parametri attuali
  - Sono trasmessi dal programma all'atto della chiamata
  - Devono corrispondere ai parametri formali in
    - ✓ Numero
    - ✓ Posizione
    - ✓ Tipo

Esempio

```
int max(int x, int y)
{
   if (x > y)
      return x;
   else
      return y;
}

main()
{
   int z = 8;
   int m;
   m = max(z, 4);
   printf("il massimo tra %d e %d è %d\n",
          z, 4, m);
}
```



Esempio

```
int max(int x, int y)
{
   if (x > y)
      return x;
   else
      return y;
}

main()
{
   int z = 8;
   int m1, m2;
   m1 = max(z, 4);
   m2 = max(7, z);
}
```

Parametri formali

All'atto di questa chiamata si effettua un legame tra:
- z e x
- 4 e y

All'atto di questa chiamata si effettua un legame tra:
- 7 e x
- z e y

```
#include <stdio.h>

double quad(float);
double cubo(float);
double quar(float);
double quin(float);
double pote(float, int);

main()
{
int base, esponente;
double ptnz;
printf(" Inserire base: " );
scanf("%d", &base);
printf(" Inserire esponente (0-5): ");
scanf("%d", &esponente);
ptnz = pote( base, esponente);
if (ptnz == -1)
printf("Potenza non prevista\n");
else
printf("La potenza %d di %d e' %f \n",
    esponente, base, ptnz);
}
```

```
double quad(float c)
{
return(c*c);
}
double cubo(float c)
{
return(c*c*c);
}
double quar(float c)
{
return(c*c*c*c);
}
double quin(float c)
{
return(c*c*c*c*c);
}
double pote(float b, int e)
{
switch (e) {
case 0: return (1);
case 1: return (b);
case 2: return (quad( b ));
case 3: return (cubo( b ));
case 4: return (quar( b ));
case 5: return (quin( b ));
default : return (-1);
}}
```

definizione della Funzione pote

Visibilità

- Una dichiarazione introduce un nome in un determinato ambito di definizione, detto *scope*. Cioè un nome è *visibile* in una specifica parte del testo del programma.
  - per un nome dichiarato all'interno del corpo di una funzione (nome *locale*), la visibilità si estende dal punto di dichiarazione alla fine del blocco in cui esso è contenuto.
  - Per un nome definito al di fuori di una funzione (nome *globale*), la visibilità si estende dal punto di dichiarazione alla fine del file in cui è contenuta la dichiarazione.

Esempio

```
int x;
f()
{
   int y;
   y = 1;
}
```

La visibilità si estende da qui fino alla fine del blocco

Anche i parametri formali di una funzione hanno un campo di visibilità che si estende dall'inizio alla fine del blocco istruzioni della funzione; sono quindi considerati a tutti gli effetti variabili locali alla funzione

Esempio

```
int x;
g(int y, char z)
{
   int k;
   int l;
   . . .
}
```

- Le variabili y e z sono locali alla funzione g e hanno un visibiltà che si estende dalla { alla }. Lo stesso dicasi per le variabili k e l all'interno del blocco.
- Per questo motivo la funzione:

```
f(int x) {
   int x;
}
```

Errata, perché tenta di definire due volte la variabile locale x nello stesso blocco

Mascheramento

- Una dichiarazione di un nome in un blocco può *mascherare* la dichiarazione dello stesso nome in un blocco più esterno o la dichiarazione dello stesso nome globale.
- Un nome ridefinito all'interno di un blocco nasconde il significato precedente di quel nome, significato che verrà ripristinato all'uscita del blocco di appartenenza.

Esempio

```
int x;          /* nome globale */
f() {
   int x;       /* x locale che nasconde x globale */
   x = l;       /* assegna 1 a x locale */
   {
      int x;    /* nasconde il primo x locale */
      x = 2;    /* assegna 2 al secondo x locale */
   }
   x = 3;       /* assegna 3 al primo x locale */
}
scanf("%d", &x);  /* inserisce un dato in x globale
```

return 1/2

- Ad ogni funzione C è associato un tipo, che caratterizza un valore.
- Questo valore è detto *valore di ritorno* della funzione ed è restituito dalla funzione al programma chiamante per mezzo della istruzione *return*.
- Sintassi:

  return (espressione) oppure *return espressione*

return 2/2

- Quando all'interno di un blocco di una funzione si incontra una istruzione return, il controllo viene restituito al programma chiamante, insieme a *espressione*. Per esempio, la funzione cubo()

```
double cubo( float c)
{
return( c*c*c );
}
```

- restituisce il controllo al programma chiamante e ritorna il cubo di c per mezzo dell'istruzione
  return (c*c*c);

Passaggio dei Parametri in C

- In C, i parametri sono trasferiti sempre e solo per valore
  - ➔ Si trasferisce *una copia* del parametro attuale, non l'originale!
  - ➔ Tale copia è strettamente privata e locale a quella funzione.
  - ➔ La funzione potrebbe quindi alterare il valore ricevuto senza che ciò abbia alcun impatto sul parametro attuale.

Esempio: Valore Assoluto

- Definizione formale:
  $|x| : Z \rightarrow N$
  $|x| = x$ se $x \geq 0$
  $|x| = -x$ se $x < 0$

- Codifica sotto forma di funzione C:

```
int ValAss(int x)
  {
    if (x < 0)
      return -x;
    else
      return x;
  }
```

Esempio: Valore Assoluto

```
int ValAss(int x)
{
  if (x < 0)
    x = -x;
  return x;
}

main()
{
  int absz, z = -87;
  absz = ValAss(z);
  printf("%d", absz);
}
```

*Quando ValAss(z) viene invocata, il valore attuale di z viene **copiato** e passato a ValAss*

ValAss riceve quindi una copia del valore -87 e la lega al simbolo x. Poi si valuta l'istruzione condizionale e si restituisce il valore 87 che viene assegnato a absz.

NOTA: Il valore di z non viene modificato

void

- Viene utilizzato da quelle funzioni che non restituiscono alcun valore e non passano parametri:
  - ➔Funzioni il cui scopo e' la visualizzazione di un messaggio o la produzione di un'uscita su uno dei dispositivi periferici.
  - ➔Vengono dette anche *sink*, cioe' funzioni "lavandino", perche' prendono dati che riversano in una qualche uscita, senza ritornare niente al chiamante.

Esempio void

```
#include <stdio.h>
      void quadrati();
main()
  {
     quadrati();
  }
void quadrati()
  {int loop;
      for (loop = 1; loop <= 10; loop++)
      printf("%d\n",loop*loop);
  }
```

```
1
4
9
16
25
36
49
64
81
100
```

E' obbligatorio mettere le parentesi () dopo il nome della funzione anche se non ci sono parametri

Esempio

Questa funzione converte un numero decimale in binario col metodo delle divisioni successive

```
void stampa_bin(int v) {
    int i, j;
    char a[DIM_INT];
    if (v==0) printf("%d", v);
    else {
          for (i=0; v!=0; i++) {
                a[i] = v%2;          /* resto della divisione tra interi */
                v/=2;
          }
          for (j= i-1; j>=0; j--)
                printf("%d", a[j]);
    }
}
```

N.B. manca il return. Viceversa, quando per una funzione non e' specificato il tipo void, per il valore di ritorno, nel blocco istruzioni della funzione e' logico che ci sia almeno una istruzione return.

void

- Viene utilizzato anche da quelle funzioni che non assumono alcun parametro:
  - ➔Es:

```
void mess_err(void) {
  int i;
  char c;
  for (i=0; i<=20; i++) printf ("\n");
  printf("   ERRORE! DENOMINATORE NULLO");
  printf("\n Premere un tasto per continuare\n");
  scanf("%c", &c);
}
```

N.B. la funzione ha soltanto il compito di visualizzare un messaggio di errore

void

- Si poteva anche scrivere:
  a. void mess_err();
  b. mess_err();

  Sono scritture equivalenti.

  ➔Nel caso b. mess_err viene considerata come una funzione il cui eventuale valore di ritorno e' di tipo int.

  Lo stesso vale anche della funzione main. L'abbiamo definita con:

  ```
  main()
  {
  ...
  }
  ```

  a indicare il fatto che non ritorna nessun valore – è quindi di tipo void – e che non assume parametri. Una equivalente (e forse anche più corretta) definizione di main potrebbe essere:

  ```
  void main( void )
  {
  …
  }
  ```