Funzioni C

a. Un programma in C è un insieme di funzioni;

b. Ogni funzione prende in ingresso un insieme di argomenti e restituisce un valore.

c. Ci deve sempre essere una funzione principale `main()`

Fino ad ora la sola funzione `main()` e *funzioni di libreria* (`#include<stdio.h>`).

Vediamo di seguito:

a. Come si definiscono le funzioni (definizione di funzione)
b. Come si usano le funzioni (chiamata o attivazione di funzione).
In realtà questo si è già visto con le funzioni di libreria.

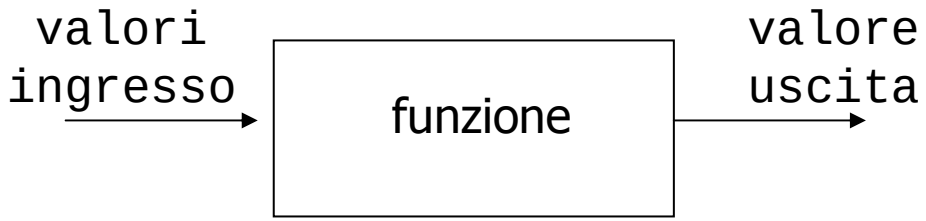
Esempio Programma con Funzione C

Programma con una funzione `square` per calcolare il quadrato di un numero.

```
#include<stdio.h>
int square(int); /*dichiarazione di funzione (o prototipo)*/

int main() {
   int x;

   for(x=1;x<=10;x++)
      printf("il numero %d quadrato di %d \n",square(x),x);

   return 0;

}
int square(int y) {/* definizione della funzione */
   return y * y;
}
```

Dichiarazione di Funzioni C

Come le variabili devono essere dichiarate prima di poter essere usate, anche le funzioni necessitano di una dichiarazione.



Sono ammesse funzioni senza uscita e/o senza ingresso

La dichiarazione di una funzione dovrà specificare:

a. Il nome della funzione: identificatore che ne permetterà l'utilizzo nel programma;

b. Gli argomenti (parametri) della funzione: quantità e tipo dei valori forniti in ingresso;

c. Il tipo del valore in uscita.

Dichiarazione di Funzioni C

Esempio: Funzione massimo tra due numeri:
- nome: `max`
- 2 argomenti di tipo `int`
- risutato: argomento di tipo `int`

Dichiarazione: `int max(int, int);`

Sintassi di dichiarazione di funzione (o prototipo):

```
identificatore-tipo identificatore(lista-tipo-parametri-formali);
```

- `identificatore-tipo` specifica il tipo del valore di ritorno (tipo del risultato della funzione), se manca viene assunto `int`

`-identificatore` specifica il nome della funzione (un qualunque identificatore C)

`-lista-tipo-parametri-formali` specifica il tipo dei dati passati alla funzione.
Lista è: `tipo1 nome1, …, tipoN nomeN` (parametri come variabili)
o solo: `tipo1, …, tipoN` (nella dichiarazione facoltativo nome parametro)

Esempi Dichiarazione di Funzioni C

Prototipi:

```
int fattoriale(int);    oppure
int fattoriale(int n);

int mcd(int, int);      oppure
int mcd(int a, int b);

double lit2euro(int);   oppure
double lit2euro(int lire);
```

Il nome del parametro può essere incluso nel prototipo per documentare meglio il programma, comunque sarà ignorato dal compilatore:

il prototipo associa il nome della funzione al suo tipo

$$\text{f: tipo}_1 \times \text{tipo}_2 \times \ldots \times \text{tipo}_n \rightarrow \text{tipo}$$

Tipo `void`

Esistono funzioni che non producono alcun effetto (es. funzione di stampa)

Il linguaggio C mette a disposizione un tipo speciale `void`

Ad es. `void f(int,int);`

è il prototipo di una funzione che non restituisce output

Es.
```
void f(int a, int b) {
        printf("%d",a*b);
      }
```

Il tipo `void` viene utilizzato anche per specificare l'assenza di argomenti:

le dichiarazioni `int f(void);` e `int f();` sono equivalenti

Definizione di Funzioni C

La definizione di una funzione specifica cosa fa una funzione e come lo fa: contiene la sequenza di istruzioni che dovrà essere eseguito per ottenere il valore risultato a partire dai dati in ingresso (una sorta di sottoprogramma).

La definizione di una funzione consta di due parti fondamentali:

1. Intestazione: simile alla dichiarazione (però necessaria la specifica dei parametri)
2. Blocco: del tutto simile al blocco già introdotto per la funzione `main()` e può contenere istruzioni e dichiarazioni di variabili (nota: non di funzioni!)

Esempi Definizione Funzioni C

Esempi Funzioni C

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

Blocco: con **`return`** viene terminata ogni funzione C

Intestazione ⇨ `int max(int x, in y)`

nome — Parametri formali

Attenzione! Se si omette il tipo di un parametro questo è assunto `int`

```
int max(double x, y)
```

equivale a: `int max(double x, int y)`

e non a: `int max(double x, double y)`

Esercizio definire una funzione convertitore euro-lire: in ingresso prende un valore in lire, in uscita determina l'equivalente in euro

Definizione Funzioni C

Sintassi definizione funzione

```
intestazione blocco
```

- `blocco` è il corpo della funzione;
- `intestazione` è l'intestazione della funzione con la forma seguente:

```
identificatore-tipo identificatore ( lista-parametri-formali )
```

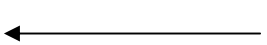
- `identificatore-tipo` specifica il tipo del valore di ritorno (tipo del risultato della funzione), se manca viene assunto `int`

`-identificatore` specifica il nome della funzione (un qualunque identificatore C)

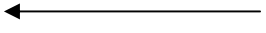
`-lista-parametri-formali` specifica i dati passati alla funzione, è una lista di dichiarazioni di parametri (tipo nome) separate da virgola, ogni parametro è una variabile (la lista può essere vuota).

Esempio Definizione Funzioni C

```
/* trovare il maggiore di tre interi */
#include <stdio.h>

int maximum(int, int, int);

int main() {
   int a, b, c;

   printf("digita tre interi: ");
   scanf("%d%d%d",&a,&b,&c);
   printf("il massimo è: %d",maximum(a,b,c));
   return 0;
}
int  maximum(int x, int y, int z){
   int max = x;

   if (y > max) max = y;
   if (z > max) max = z;
   return max;
}
```



intestazione



blocco

Attivazione di Funzioni

Le funzioni vengono chiamate (attivate) all'interno di funzioni

```
. . .
int valore, x, y;
  . . .
x = 1; y = 2;
valore = max(x,y);
  . . .
```

Sintassi Attivazione Funzione

```
identificatore (lista-parametri-attuali)
```

- `identificatore` è il nome della funzione
- `lista-parametri-attuali` è una lista di espressioni separate da virgola.

I parametri attuali devono corrispondere in numero e tipo ai paramenti formali.

Coercizione argomenti

Una chiamata di funzione che non corrisponda al prototipo provocherà un errore di sintassi.

Es.
```
int x;
x = maximum(1,3);
```

Errore a tempo di compilazione

In alcuni casi tollerate chiamate con tipi diversi: *coercizione degli argomenti*.

Es. `sqrt` ha come argomento un `double`, però:
`int x = 4;`
`printf("%f",sqrt(x));` stampa `2.00…0`
Il compilatore *promuove* il valore intero `4` al valore `double 4,0`
quindi la funzione viene chiamata correttamente.

Le conversioni avvengono secondo le *regole di promozione del C*

Attivazione di Funzioni

Semantica Attivazione Funzione di una funzione B in una funzione A

1. L'attivazione di una funzione è un'espressione.
2. L'attivazione di B da A determina:
    a. viene sospesa l'esecuzione di A e si passa ad eseguire le istruzioni di B (a partire dalla prima)
    b. quando termina l'esecuzione di B si torna ad eseguire le istruzioni di A

Prima di poter essere attivata una funzione deve essere definita (o dichiarata)

Quindi definita o dichiarata sopra.

Dove si definiscono le funzioni?

In C non si possono definire funzioni dentro funzioni.

Quindi fuori dalla funzione `main()` (sopra o sotto)

Esempio:

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

int main {
   int a, b,c;

   scanf("%d, %d",&a,&b);
   max = max(a,b);
   return max;
}
```

definizione di funzione

chiamata di funzione

La funzione `max` è visibile nel `main`

Visibilità delle Funzioni

Prima di essere attivata una funzione deve essere definita o dichiarata:
il compilatore deve controllare che i parametri formali corrispondano (per numero e tipo) ai parametri attuali.

prima dell'attivazione deve essere conosciuto il prototipo

Esempio:

```
int main() {
   int a, b,c;

   scanf("%d%d",&a,&b);
   max = max(a,b);
   prinf("il max è %d",c);
   return 0;
}
int max(int x, int y) {
   if (x >= y) return x;
   else return y;
}
```

`max` non è visibile ed il compilatore C non può controllare i parametri

Visibilità delle Funzioni

Se la funzione non è definita almeno deve essere già dichiarata

```
int max(int, int);
int main () {
   int a, b,c;

   scanf("%d%d",&a,&b);
   c = max(a,b);
   prinf("il max è %d",c);
   return 0;
}
int max(int x, int y) {
   if (x >= y) return x;
   else return y
}
```


dichiarazione di funzione

Ordine dichiarazione delle funzioni

Ogni funzione deve essere stata dichiarata prima di essere usata.

E' pratica comune specificare in quest'ordine:

1. Dichiarazione di tutte le funzioni con esclusione del `main`
2. Definizione di `main`
3. Definizione di tutte le altre funzioni

In questo modo ogni funzione e' stata dichiarata prima di essere usata, e l'ordine diventa irrilevante!

Esempio:

```
int max(int,int);
int mcm(int,int);
int main(){ …}
int max(int a, int b){ … }
int mcm(int a, int b){…}
```

Nota: i file di intestazione della lib. standard (e.g. <stdio.h>) contengono i prototipi delle funzioni.

File header (o intestazioni)

Ogni libreria standard ha un corrispondente file header che contiene:

- definizioni di costanti
- definizioni di tipo
- dichiarazioni di tutte le funzioni della libreria

Esempi:

`<stdio.h>`	input/output
`<stdlib.h>`	memoria, numeri casuali, utilità generali
`<string.h>`	manipolazione stringhe
`<limits.h>`	limiti del sistema per valori interi
`<float.h>`	limiti del sistema per valori reali
`<math.h>`	funzioni matematiche

…

Possono essere scritte dal programmatore “`mialib.h`”

Funzioni della libreria matematica

Per utilizzarle occorre: `#include<math.h>`

Sia argomenti che valore di ritorno reali in doppia precisione, ovvero tipo `double` (non `float`)

Funzioni disponibili:

`sqrt(x)`	radice quadrata
`exp(x)`	e^x
`log(x)`	logaritmo naturale
`log10(x)`	logaritmo base 10
`fabs(x)`	valore assoluto
`ceils(x)`	arrotonda all'intero più piccolo > x
`floor(x)`	arrotonda all'intero più grande < x
`pow(x,y)`	x^y
`fmod(x,y)`	resto di x/y (in virgola mobile)
`sin(x), cos(x), tan(x)`	trigonometriche (x in radianti)