

C: Funzioni

Le funzioni

Le funzioni sono dei **sottoprogrammi** costituiti da un insieme di istruzioni

Il loro uso permette di realizzare programmi più chiari da leggere, da capire e da modificare, infatti...

- attraverso l'uso delle funzioni si evita di dover riscrivere più volte del codice
- è possibile chiamare una funzione più volte in un programma

Le funzioni sono alla base della *programmazione modulare*

Creazione librerie (scanf, printf) che nascondono i dettagli implementativi

Funzioni: introduzione

valori in ingresso -> FUNZIONE -> valore restituito

Abbiamo già usato le funzioni!

```
int main() {  
    return 0;  
}
```

Funzioni: dichiarazione

1. scelta del **nome** della funzione
2. **tipo** del valore restituito
3. **tipo** (e nome) dei parametri in ingresso (*parametri formali*)

```
void nome_funzione(void);
```

oppure

```
int somma(int, int);
```

Funzioni: definizione

Con la definizione di una funzione *specifichiamo* le istruzioni che caratterizzano la funzione

Esempio:

```
int somma(int c,int d) //nome dei parametri formali
{
    int s;
    s=c+d;
    return s; //restituiamo s al main (o alla funzione chiamante)
}
```

La definizione deve essere inserita subito dopo la chiusura del
`main`

Funzioni: chiamata

La **chiamata** permette di *eseguire* la funzione, passare i parametri (*attuali*) ed assegnare il valore restituito

Esempio:

```
int main() {  
    int a=1,b=1,sum;  
  
    sum=somma(a,b); //a e b sono i parametri attuali  
    printf("\nLa somma vale %d\n",sum);  
}
```

👉 Nella chiamata occorre fare attenzione che il **tipo**, il **numero** e l'**ordine** dei paramtri attuali sia conforme ai parametri formali indicati nella definizione della funzione! **!**

Funzioni: primo esempio

```
#include <stdio.h>

void messaggio();

int main() {
    messaggio();
}

void messaggio() {
    printf("\nLa mia prima funzione\n");
}
```

Funzioni: secondo esempio

```
#include <stdio.h>

void messaggio();

int main() {
    int i;

    for(i=1; i<=5; i++)
        messaggio();
}

void messaggio() {
    printf("\nLa mia prima funzione\n");
}
```


Funzioni: altro esempio

```
#include <stdio.h>

void pari_dispari(int);

int main() {
    int num;

    printf("\nInserisci un numero intero ");
    scanf("%d",&num);
    pari_dispari(num);
}

void pari_dispari(int n) {

    if(n%2==0) printf("\nIl num e' pari\n");
    else printf("\nIl num e' dispari\n");
}
```

Funzioni: utilizzo di più parametri

```
#include <stdio.h>

void bmi(float, float);

int main() {
    float peso, altezza;

    printf("\nPeso in Kg: ");
    scanf("%f", &peso);
    printf("\nAltezza in m: ");
    scanf("%f", &altezza);
    bmi(peso, altezza);
}

void bmi(float p, float a) {
    printf("\nIl bmi vale %.2f\n", p/(a*a));
}
```

Funzioni: utilizzo di più parametri e return

```
#include <stdio.h>

float bmi(float, float);

int main() {
    float peso, altezza, mio_bmi;

    printf("\nPeso in Kg: ");
    scanf("%f", &peso);
    printf("\nAltezza in m: ");
    scanf("%f", &altezza);
    mio_bmi = bmi(peso, altezza);
    printf("\nIl bmi vale %.2f\n", mio_bmi);
}

float bmi(float p, float a) {
    float ris;

    ris = p / (a * a);
    return ris;
}
```

Funzioni: utilizzo di più parametri e return

Oppure...

```
#include <stdio.h>

float bmi(float, float);

int main() {
    float peso, altezza, mio_bmi;

    printf("\nPeso in Kg: ");
    scanf("%f", &peso);
    printf("\nAltezza in m: ");
    scanf("%f", &altezza);
    mio_bmi = bmi(peso, altezza);
    printf("\nIl bmi vale %.2f\n", mio_bmi);
}

float bmi(float p, float a) {
    return p / (a * a);
}
```

Funzioni: passaggio dei parametri

👉 In C avviene sempre e solo **per valore**

- *i parametri formali vengono inizializzati con i corrispondenti valori dei parametri attuali*

Una funzione **non modifica** i valori dei parametri attuali 

Funzioni: passaggio dei parametri

La variabile `num` non viene modificata dalla funzione

```
#include <stdio.h>

int quadrato(int);

int main() {
    int num,q;

    printf("\nInserisci un numero intero: ");
    scanf("%d",&num);
    q=quadrato(num); //num non viene modificato
    printf("\nIl quadrato di %d vale %d\n",num,q);
}

int quadrato(int n) {
    n=n*n;
    return n;
}
```

Funzioni: passaggio dei parametri

La variabile `num` non viene modificata dalla funzione

```
#include <stdio.h>

int quadrato(int);

int main() {
    int num,q;

    printf("\nInserisci un numero intero: ");
    scanf("%d",&num);
    q=quadrato(num);
    printf("\nIl quadrato di %d vale %d\n",num,q);
}

int quadrato(int num) { //neanche in questo caso!!!
    num=num*num;
    return num;
}
```

Funzioni: passaggio dei parametri

Passaggio **esplicito**: utilizzo dei parametri attuali e formali

Passaggio **implicito**: utilizzi di variabili globali (è buona norma evitarle!)

Visibilità (scope)

☞ La dichiarazione di una variabile introduce un nome simbolico **visibile** (scope) all'interno di una specifica parte (blocco) del programma

Nome locale – **variabile locale**

- visibilità solo all'interno del blocco delle istruzioni della funzione: dal punto della dichiarazione alla fine del blocco

Nome globale – **variabile globale**

- definita fuori dalla funzione: dal punto della dichiarazione alla fine del file

☞ I parametri formali sono delle variabili locali!

Visibilità (scope)

Esempio:

```
#include <stdio.h>

int main() {

    for(int i=0; i<10; i++)
        printf("%d", i);
    printf("%d", i);

}
```

Cosa produce questo programma?

Visibilità (scope)

Esempio:

```
#include <stdio.h>

int main() {
    int i;

    for(i=0; i<10; i++)
        printf("%d", i);
    printf("%d", i);
}
```

E questo?

Visibilità (scope)

Occorre prestare particolare attenzione al **mascheramento**

Mascheramento: la dichiarazione di una variabile locale può *nascondere* (e quindi *mascherare*) una variabile con lo **stesso nome** dichiarata in un blocco più esterno (o globale)

Mascheramento

```
#include <stdio.h>

int main() {
    int i=-1;

    for(int i=0; i<10; i++)
        printf("%d ", i);
    printf("\n%d", i);
}
```

Provate questo codice

Mascheramento

```
#include <stdio.h>

int quadrato(int);
int n=4; //variabile globale

int main() {
    int n=3; //ATTENZIONE al mascheramento

    printf("\nIl quadrato di %d vale %d\n",n,quadrato(n));
    //Il quadrato di 3 vale 9
}

int quadrato(int n) {
    n=n*n;
    return n;
}
```

E' possibile che il mascheramento avvenga senza rendersene conto!

Scope: variabile globale

```
#include <stdio.h>

int quadrato(int);
int n=4; //variabile globale

int main() {

    printf("\nIl quadrato di %d vale %d\n",n,quadrato(n));
    //Il quadrato di 4 vale 16
    printf("Ora n vale: %d\n",n);
    //Ora n vale 4
}

int quadrato(int n) {
    n=n*n;
    return n;
}
```

Scope: variabile globale

```
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {

    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di 4 vale 16
    printf("Ora n vale: %d\n",n);
    //Quanto vale n?
}

int quadrato() {
    n=n*n;
    return n;
}
```


Scope: variabile globale

```
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {

    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di 4 vale 16
    printf("Ora n vale: %d",n);
    //Ora n vale 16 <-----
}

int quadrato() {
    n=n*n;
    return n;
}
```

Scope: e ancora...

```
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {
    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di 4 vale 16
    printf("Ora n vale: %d\n",n); //Ora n vale 16
    int n=5;
    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di ? vale ?
    printf("Ora n vale: %d\n",n); //Ora n vale ?
}

int quadrato() {
    n=n*n;
    return n;
}
```

Scope: e ancora...

```
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {

    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di 4 vale 16
    printf("Ora n vale: %d\n",n); //Ora n vale 16
    int n=5;
    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di 5 vale 25
    printf("Ora n vale: %d\n",n); //Ora n vale 5
}

int quadrato() {
    n=n*n;
    return n;
}
```

Scope: ultimo esempio

```
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {
    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di 4 vale 16
    printf("Ora n vale: %d\n",n);
    //Ora n vale 16
    n=5;
    printf("\nIl quadrato di %d vale %d\n",n,quadrato());
    //Il quadrato di 5 vale 25
    printf("Ora n vale: %d\n",n);
    //Ora n vale 25
}

int quadrato() {
    n=n*n;
    return n;
}
```

Passaggio dei parametri

Ricapitolando...

- **Parametri formali:** definizione della funzione (tipo, numero e ordine)
- **Parametri attuali:** passati alla funzione quando viene chiamata

👉 In C il passaggio dei parametri avviene per valore, cioè durante la chiamata di una funzione ogni parametro formale viene inizializzato con il valore del corrispondente parametro attuale

💡 E se volessimo modificare il valore dei parametri attuali all'interno della funzione?

Passaggio dei parametri

💡 E se volessimo modificare il valore dei parametri attuali all'interno della funzione?

☀️ Possiamo passare per valore l'indirizzo di una variabile, cioè inizializzare i parametri formali con l'indirizzo dei parametri attuali.

E quindi usare i **puntatori**!

Esempio: scambia

```
#include <stdio.h>
void scambia(int, int);

int main() {
    int x=0,y=1;
    printf("\nPrima x vale %d e y vale %d",x,y);
    scambia(x, y);
    printf("\nDopo x vale %d e y vale %d\n",x,y);
}

void scambia(int x, int y) {
    int tmp;
    tmp=x;
    x=y;
    y=tmp;
}
```

Esempio: scambia con i puntatori

```
#include <stdio.h>

void scambia(int *, int *);

int main() {
    int x=0,y=1;
    printf("\nPrima x vale %d e y vale %d",x,y);
    scambia(&x, &y);
    printf("\nDopo x vale %d e y vale %d\n",x,y);
}

void scambia(int *x, int *y) {
    int tmp;
    tmp=*x;
    *x=*y;
    *y=tmp;
}
```

Prima x vale 0 e y vale 1 e Dopo x vale 1 e y vale 0