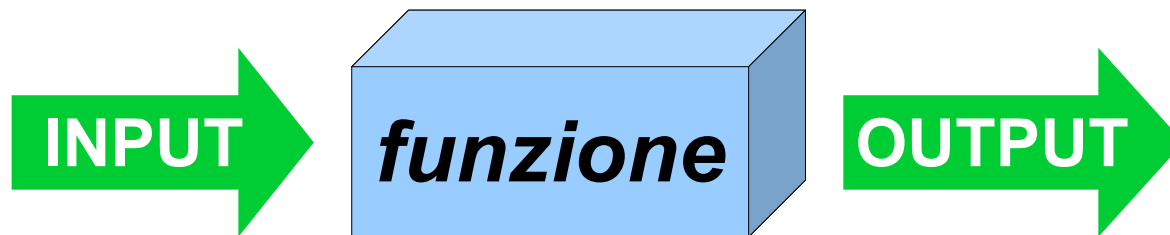


Funzioni

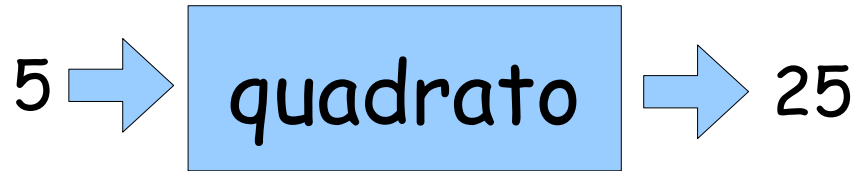
Una funzione è una "regola" che indica come trasformare i dati di ingresso (input) in modo da produrre un risultato in uscita (output).



*i valori di input si dicono
argomenti (o parametri)
della funzione*

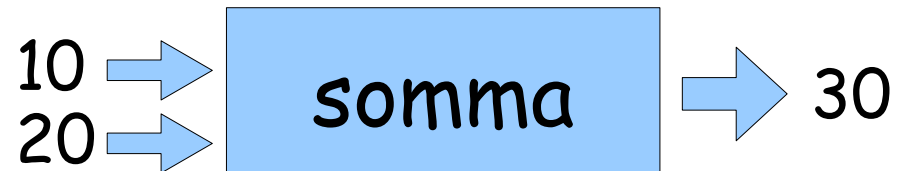
*OUTPUT = valore
restituito dalla
funzione
(return value)*

Esempi di funzioni



Esempio 1. Funzione che calcola il quadrato di un numero

Esempio 2. Funzione che riceve in ingresso due numeri e ne calcola la somma





Esempio 3. Funzione che riceve in ingresso una sequenza di caratteri (stringa) e la trasforma in maiuscolo

Esempio 4. Funzione che riceve in ingresso 3 numeri e restituisce il minimo






Analogia con il foglio di calcolo

Esempio 1 la funzione RADQ in calc riceve come argomento un numero e restituisce la sua radice quadrata

C3						=	=RADQ(B3)
	A	B	C				
1							
2		INPUT	OUTPUT				
3		2	1,4142135624				
4		3	1,7320508076				
5		4	2				
6		5	2,2360679775				
7		6	2,4494897428				
8							

Analogia con il foglio di calcolo

Esempio 1 la funzione RESTO in calc riceve come argomenti due numeri interi e restituisce il resto della divisione tra il primo e il secondo

D3    = <input type="text" value="=RESTO(B3;C3)"/>				
	A	B	C	D
1				
2		INPUT1	INPUT2	OUTPUT
3		10	2	0
4		10	3	1
5		10	4	2
6		10	5	0
7		10	6	4
8		10	11	10
9				

Funzioni predefinite in C

In C è disponibile la libreria `math.h` che contiene numerose funzioni matematiche predefinite. Esempio:

```
#include <math.h>
```

```
int main() {  
    float x = 2, y=3, z, t;  
    z = sqrt( x ); // radice  
    t = pow( 0.5 , y ); // potenza  
    printf( "z = %f t = %f", z , t );  
}
```

 C:\Users\Me\Desktop\Untitled1.exe

z = 1.414214 t = 0.125000

Process exited after 0.01204 seconds
Premere un tasto per continuare

Definizione di una funzione

- Oltre ad utilizzare le funzioni predefinite, in C il programmatore può **definire nuove funzioni**
- Esempio: vogliamo definire la funzione **fattoriale(...)** che riceve come argomento un numero intero, e restituisce il suo fattoriale.

Nota: il fattoriale di un numero intero è il prodotto di tutti i numeri interi compresi tra 1 e quel numero.

Esempio: il fattoriale di 5 è $1*2*3*4*5 = 120$

Definizione di un funzione

```
#include <stdio.h>
```

```
int fattoriale( int );
```

← **PROTOTIPO della funzione fattoriale**

```
int main() {
```

```
    int a=1, b=2, x, y;
```

← **VARIABILI LOCALI della funzione main**

```
    x = fattoriale( 5 );
```

```
    y = fattoriale( a+b );
```

```
    printf("%d %d", x,y);
```

```
    system("PAUSE");
```

```
}
```

```
int fattoriale( int num) {
```

```
    int i, fatt=1;
```

← **VARIABILI LOCALI della funzione fattoriale**

```
    for(i=1;i<=num;i++) {
```

```
        fatt*=i;
```

```
    }
```

```
    return fatt;
```

```
}
```

← **Due INVOCAZIONI (CHIAMATE) della funzione fattoriale da parte della funzione main**

NOTA BENE: quando si incontra la parola chiave return, la funzione termina immediatamente e restituisce il valore alla funzione chiamante (in questo caso la funzione main).

Esempio 2: funzione "2x-y"

tipo di ritorno

float miaFunz(float, float);

numero e tipo parametri

```
int main() {
```

```
    float a = 3, b = 4, c, d;
```

```
    c = miaFunz( a, b );
```

```
    d = miaFunz( b, a );
```

```
    c = miaFunz( a );
```

```
    printf("%f %f", c, d);
```

```
}
```

PARAMETRI ATTUALI

i parametri attuali
devono **SEMPRE**
corrispondere
in numero e tipo
ai parametri formali

```
float miaFunz( float x, float y) {
```

```
    return 2*x - y;
```

```
}
```

PARAMETRI FORMALI

RAM

a	3
b	4
c	?
d	?
...	
...	
...	

Due considerazioni importanti

```
float miaFunz( float, float );
```

```
int main() {  
    float x = 3, y = 4, c;  
    c = miaFunz( y, x );  
}
```

```
float miaFunz( float x, float y) {  
    float ris = 2*x - y;  
    x = 0;  
    y = 0;  
    return ris;  
}
```

1) i parametri attuali possono avere lo stesso nome dei parametri formali, MA SAREBBERO COMUNQUE VARIABILI DIVERSE e l'associazione avviene in base all'ordine!

2) La funzione può modificare x e y, MA LA MODIFICA NON HA ALCUN EFFETTO sulle variabili x e y del main!

RAM	
x	3
y	4
c	5
..	
...	
x	4
y	3

Riassumendo...

1. L'associazione tra parametri attuali e parametri formali avviene sempre **IN BASE ALL'ORDINE** con cui i parametri sono passati alla funzione, mai in base al nome dei parametri!
2. La funzione può assegnare un valore ai propri parametri formali, ma la modifica **NON HA EFFETTO** sulle variabili locali della funzione chiamante.

Composizione di funzioni

Una funzione può essere usata come argomento di un'altra funzione, evitando l'uso di variabili intermedie.

```
int square(int x) {  
    return x*x;  
}  
  
int sum(int x, int y) {  
    return x+y;  
}
```

```
int main() {  
    int a = 2, b = 3, c, d;  
  
    ↪ 13  
    c = sum( square(a), square(b) );  
  
    ↪ 25  
    d = square( sum(a,b) );  
}
```

Esempio: la funzione min

```
int min(int a, int b) {  
    if ( a < b ) {  
        return a;  
    }  
    else {  
        return b;  
    }  
}
```

o anche ...

```
int min(int a, int b) {  
    if ( a < b ) {  
        return a;  
    }  
    return b;  
}
```

```
int main() {  
    ....  
    m = min( min(a,b), c ); // minimo tra 3 valori  
}
```

Esempio: la funzione valore assoluto

```
int valAssoluto(int x) {  
    int risultato;  
    if ( x >= 0 ) {  
        risultato = x;  
    }  
    else {  
        risultato = - x;  
    }  
    return risultato;  
}
```

oppure ...

```
int valAssoluto(int x) {  
    if ( x >= 0 )  
        return x;  
    else  
        return -x;  
}
```

o anche...

```
int valAssoluto(int x) {  
    if ( x >= 0 )  
        return x;  
    return -x;  
}
```

Esercizio

Qual è l'output di questo programma?

```
int alfa(int a) {  
    if (a>10)  
        return a;  
    else  
        return 6;  
}
```

```
int beta(int x, int y) {  
    return x+2*y;  
}
```

```
int main() {  
    int a=0,b=0,c=0,d=0;  
    a = alfa(11);  
    b = alfa( beta(4,2) );  
    c = beta( alfa(1) , alfa(3) );  
d = alfa( beta(1,1), beta(2,2) );
```

```
    printf(" a = %d \n", a);  
    printf(" b = %d \n", b);  
    printf(" c = %d \n", c);  
    printf(" d = %d \n", a);  
}
```

SOLUZIONE:

a=11

b=6

c=18

Esercizio

Rimuovi la riga del main che produce errore di compilazione e determina l'output del programma

```
int alfa(int a) {  
    if (a>10)  
        return a;  
    else  
        return 6;  
}
```

```
int beta(int x, int y) {  
    return x+2*y;  
}
```

```
int main() {  
    int a=0,b=0,c=0,d=0;  
    a = alfa(11);  
    b = alfa( beta(4,2) );  
    c = beta( alfa(1) , alfa(3) );  
    d = alfa( beta(1,1), beta(2,2) );  
  
    printf(" a = %d \n", a);  
    printf(" b = %d \n", b);  
    printf(" c = %d \n", c);  
    printf(" d = %d \n", d);  
}
```

Esercizi per casa

1. Scrivi un programma che definisce e utilizza la *valoreAssoluto*, che riceve in ingresso un float e restituisce il suo valore assoluto.
2. Scrivi un programma che definisce e utilizza la funzione *potenza*, che riceve come argomenti base ed esponente interi e restituisce il valore della potenza.
3. Dopo aver rimosso le eventuali righe che generano errori di compilazione, determina l'output del seguente programma:

<pre>int f1(int a, int b){ if (a>b) return a; else return 2*b; } int f2(int x){ if (x>0) return x; else return -x; }</pre>	<pre>int main(){ int a=0,b=0,c=0,d=0; a = f2(5); b = f1(3 , f2(-4)); c = f2(f1(4), f1(3)); d = f2(f1(-10,-11)); printf(" a = %d \n", a); printf(" b = %d \n", b); printf(" c = %d \n", c); printf(" d = %d \n", d); }</pre>
---	--

[SOLUZIONE: a=5, b=8, c=0; d=10]

Perchè usare le funzioni?

- Evitare duplicazione di codice nel programma (se c'è un errore lo si corregge una volta sola!)
- Riutilizzare il codice già scritto (e testato) in passato per creare nuovi programmi
- Rendere PIU' LEGGIBILI i programmi

Esempio: ricerca dei numeri primi

```
/* funzione che restituisce  
   1 se il numeo passato è primo,  
   0 se il numero NON è primo. */
```

```
int primo (int n) {  
    int i;  
    for( i=2 ; i <= n/2 ; i++ ) {  
        if(n%i==0){  
            return 0;  
        }  
    }  
    return 1;  
}
```

```
int main(){  
    int c, max;  
    printf("Inserisci un valore ");  
    scanf("%d", &max);  
    for( c=2; c<max; c++) {  
        if ( primo(c) == 1 ) {  
            printf(" %d\n", c );  
        }  
    }  
}
```

si può scrivere
semplicemente:
`if (primo(c))`

Esempio: ricerca dei numeri primi

```
/* funzione che restituisce  
   1 se il numeo passato è primo,  
   0 se il numero NON è primo. */
```

```
int primo (int n) {  
    int i;  
    for( i=2 ; i <= n/2 ; i++ ) {  
        if(n%i==0){  
            return 0;  
        }  
    }  
    return 1;  
}
```

```
int main(){  
    int c, max;  
    printf("Inserisci un valore ");  
    scanf("%d", &max);  
    for( c=2; c<max; c++) {  
        if ( primo(c) == 1 ) {  
            printf(" %d\n", c );  
        }  
    }  
}
```

si può scrivere
semplicemente:
`if (primo(c))`

Esempio: ricerca dei numeri primi

```
/* funzione che restituisce  
   1 se il numeo passato è primo,  
   0 se il numero NON è primo. */
```

```
int primo (int n) {  
    int i;  
    for( i=2 ; i <= n/2 ; i++ ) {  
        if(n%i==0){  
            return 0;  
        }  
    }  
    return 1;  
}
```

```
int main(){  
    int c, max;  
    printf("Inserisci un valore ");  
    scanf("%d", &max);  
    for( c=2; c<max; c++) {  
        if ( primo(c) == 1 ) {  
            printf(" %d\n", c );  
        }  
    }  
}
```

si può scrivere
semplicemente:
`if (primo(c))`

Funzioni con tipo di ritorno "void"

```
void barra(int);
```

← prototipo di una funzione
che riceve in ingresso un
int e NON restituisce
alcun valore.

```
void saluta();
```

```
int main() {  
    saluta();  
    barra(4);  
    barra(8);  
    saluta();  
}
```

```
void barra(int lunghezza) {  
    for( int i=0; i<lunghezza;i++)  
        printf("*");  
    printf("\n");  
}
```

```
void saluta() {  
    printf("ciao\n");  
}
```

← attenzione: questa
istruzione è fuori dal
ciclo *for*

[apri esempio](#)

OUTPUT

```
ciao  
****  
*****  
ciao
```

Variabili globali e locali

```
#include <stdio.h>
```

```
void test();
```

```
int x;
```

x è una VARIABILE GLOBALE (perchè dichiarata all'esterno di ogni funz.)
x è VISIBILE (utilizzabile) da qualsiasi funzione del programma.

```
int main() {
```

```
int y;
```

y è una VARIABILE LOCALE della funzione *main*, quindi
y può essere utilizzata solo all'interno della funzione *main*

```
x = 1;
```

```
y = 2;
```

```
z = 3;
```

```
}
```

```
void test() {
```

```
int z;
```

y è una VARIABILE LOCALE della funzione *test*
y può essere utilizzata solo all'interno della funzione *test*

```
x = 100;
```

```
y = 200;
```

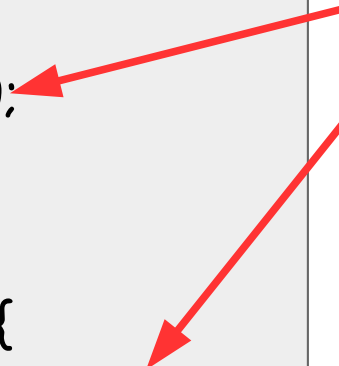
```
z = 300;
```

```
}
```

Variabili locali e globali

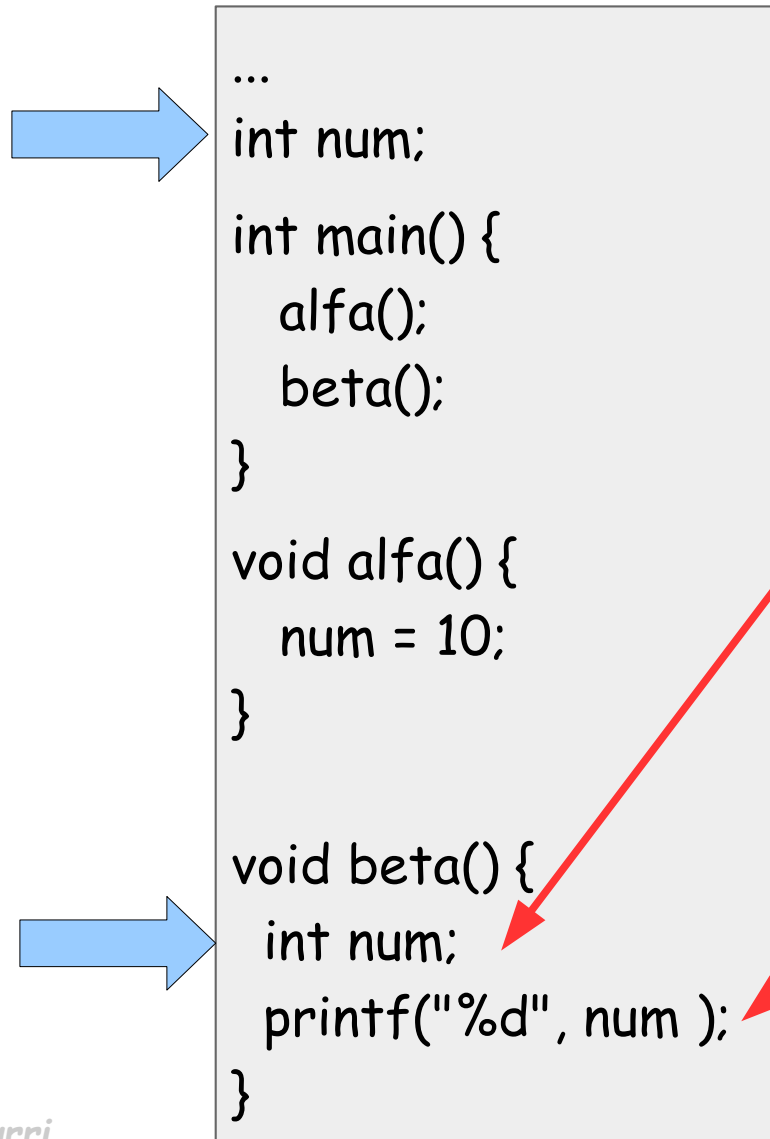
```
...  
int num = 1;  
  
int main() {  
    alfa();  
    beta();  
}  
  
void alfa() {  
    num = 100;  
}  
  
void beta() {  
    printf("%d", num );  
}
```

la modifica effettuata
dalla funzione *alfa* alla
variabile *num* è visibile
nella funzione *beta*



Mascheramento (shadowing) di una variabile

Cosa accade se una funzione dichiara una variabile locale avente lo stesso nome di una variabile globale ??



```
...
int num;

int main() {
    alfa();
    beta();
}

void alfa() {
    num = 10;
}

void beta() {
    int num;
    printf("%d", num );
}
```

La variabile locale **MASCHERA** la variabile globale con lo stesso nome, cioè la funzione "vede" solamente la variabile locale

quindi in questo viene stampato il contenuto della variabile locale num (un numero casuale!)

Esercizio: numeri perfetti

Un numero intero si dice *perfetto* se è uguale alla somma dei suoi divisori (non considerando il numero stesso).

Esempio:

28 è perfetto perchè i suoi divisori sono 1, 2, 4, 7 e 14
($1 + 2 + 4 + 7 + 14 = 28$)

Scrivi una funzione che riceve un numero intero e restituisce 1 se il numero è perfetto, 0 altrimenti. Il prototipo sarà:

int perfetto(int n);

Usa la funzione per cercare (e stampare) tutti i numeri perfetti compresi in un intervallo stabilito dall'utente.

NOTA: Esistono solo cinque numeri perfetti minori di 1.000.000.000.

Esercizio: stampa dei divisori

Scrivi un programma che richiede all'utente un numero intero positivo n compreso tra 1 e 1000, e visualizza tutti i suoi divisori.

Il programma deve utilizzare le seguenti funzioni:

```
int chiediNumero();
```

```
void stampaDivisori(int n);
```