



**UNIVERSITÀ  
DI PARMA**

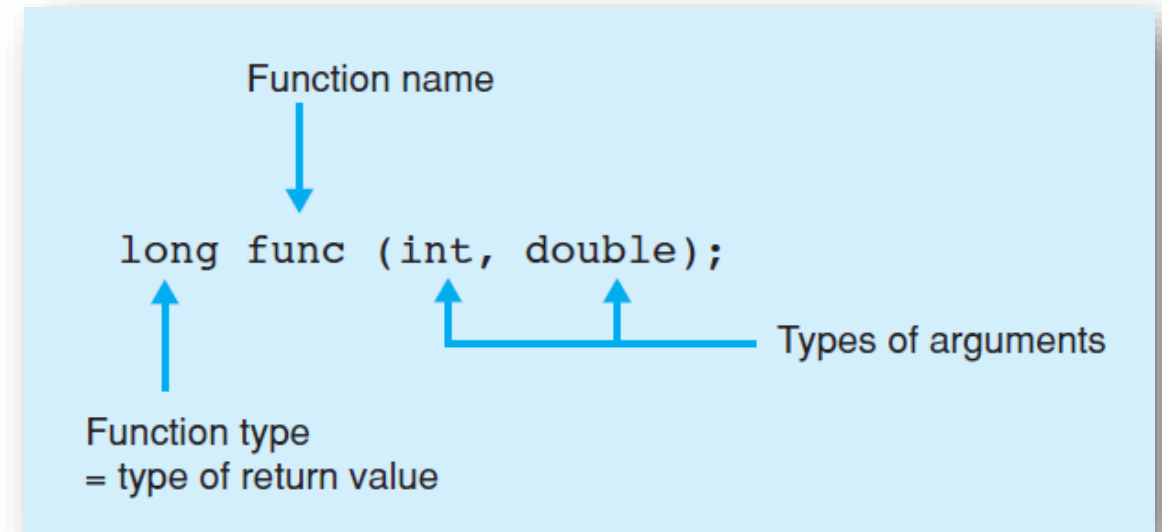
# **C++ funzioni**

## **Alberto Ferrari**

<http://en.cppreference.com>

# FUNZIONI

- caratterizzate da *nome*, *parametri* (numero, ordine e tipo) e *tipo* di ritorno
- le funzioni hanno un prototipo
- il prototipo non è necessario se la definizione della funzione appare prima del suo utilizzo
- nel prototipo i parametri possono non avere nome, ma per chiarezza in genere lo si mette



- la direttiva **#include** permette di importare i prototipi di funzioni delle librerie standard
- ogni libreria standard ha un file *header* contenente la definizione di funzioni, di tipo di dati e di costanti
- **#include <cmath>**
  - per utilizzare funzioni matematiche

```
double sin (double);           // Sine
double cos (double);           // Cosine
double tan (double);           // Tangent
double atan (double);          // Arc tangent
double cosh (double);          // Hyperbolic Cosine
double sqrt (double);          // Square Root
double pow (double, double);    // Power
double exp (double);           // Exponential Function
double log (double);           // Natural Logarithm
double log10 (double);         // Base-ten Logarithm
```

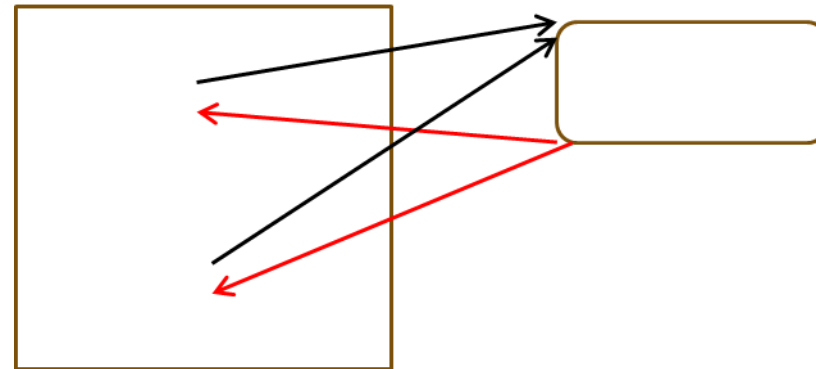
```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
    double y, x = 5.22;
    // la funzione pow ha prototipo double pow( double, double)
    // y = pow("x", 3.0); // error: no matching function
                          // for call to 'pow(const char [2], double)'
    // y = pow(x + 3.0); // error: no matching function for call to 'pow(double)'
    y = pow(x, 3.0);      // ok!
    y = pow(x, 3);        // ok! Il compilatore converte l'intero 3 in double
    cout << x << " elevato al cubo vale: " << y << endl;
    return 0;
}
```

- **prototipo** (firma) di una funzione
  - `<tipo-funzione> <nome-funzione>(<tipo-1>,<tipo-2>,...) ;`
  - `<tipo-funzione>` è il tipo del valore restituito dalla funzione
    - *void* se la funzione non restituisce valori
  - `<nome-funzione>` è il nome (*identificatore*) della funzione
  - `<tipo-1> <tipo-2>` sono i tipi dei *parametri*
    - possono mancare in funzioni senza parametri
    - oltre al tipo è possibile specificare il nome del parametro
- **definizione** di una funzione
  - analogo al prototipo
  - è obbligatorio specificare i *nomi* dei parametri (*formali*)
  - è obbligatorio specificare il *corpo* della funzione
  - è obbligatorio specificare il valore di ritorno mediante l'istruzione *return* (*non necessario per funzioni void*)

- ***chiamata*** della funzione (provoca la sua ***esecuzione***)
  - ***nome*** della funzione e ***lista dei parametri attuali***
  - è possibile inserire più istruzioni di chiamata funzione
  - specificare l'uso del valore di ritorno (per funzioni non void)



```
#include <iostream>
using namespace std;
double media(int v1, int v2);
int main() {
    int val1, val2;
    cout << "Inserire due valori interi separati da spazio ";
    cin >> val1 >> val2;
    cout << "la media aritmetica fra " << val1 << " e " << val2
          << " = " << media(val1,val2) << endl;

    return 0;
}
double media(int v1, int v2) {
    double med;
    med = (v1 + v2) / 2.0;
    return med;
}
```



- ***call-by-value***
  - il parametro attuale può essere una ***variabile o un'espressione***
  - il parametro formale viene inizializzato al ***valore*** del corrispondente parametro attuale
  - la funzione riceve una ***copia del valore*** del parametro
  - le azioni sui parametri formali ***non si ripercuotono*** sui parametri attuali
- ***call-by-reference*** (“&” precede il tipo del parametro formale)
  - il parametro attuale deve essere una ***variabile***
  - il parametro formale viene associato al parametro attuale (si riferisco alla stessa zona di memoria)
  - le azioni sui parametri ***formali si ripercuotono*** sui parametri attuali
  - ***vantaggio***: migliori ***prestazioni***
  - ***svantaggio***: minore modularità, la funzione chiamata può corrompere i dati della chiamante (***side effects***)

- *call-by-pointer*

- l'*indirizzo* del parametro attuale viene copiato nel parametro formale
- all'interno della funzione l'indirizzo è utilizzato per accedere al dato "puntato" dal parametron attuale
- i *parametri formali* devono essere definiti come *puntatori* ai quali assegnare gli *indirizzi* dei parametri *attuali*
- le azioni sui parametri formali *si ripercuotono* sui parametri attuali

```
void scambiaVal(int a, int b) {  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
}  
  
int main() {  
    ...  
    int x,y; x = 7; y = 5;  
    cout << "scambio i valori x = " << x << " y = " << y << endl;  
    cout << "passaggio per valore : ";  
    scambiaVal(x,y);  
    cout << "x = " << x << " y = " << y << endl;  
    ...  
}
```

```
scambio i valori x = 7 y = 5  
passaggio per valore : x = 7 y = 5
```

```
void scambiaRef(int &a, int &b) {  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
}  
  
int main() {  
    ...  
    cout << "scambio i valori x = " << x << " y = " << y << endl;  
    cout << "passaggio per riferimento : ";  
    scambiaRef(x,y);  
    cout << "x = " << x << " y = " << y << endl; ...  
}
```

```
scambio i valori x = 7 y = 5  
passaggio per riferimento : x = 5 y = 7
```

```
void scambiaInd(int *a, int *b) {  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main() {  
    ...  
    cout << "scambio i valori x = " << x << " y = " << y << endl;  
    cout << "passaggio per indirizzo : ";  
    scambiaInd(&x, &y);  
    cout << "x = " << x << " y = " << y << endl;  
    ...  
}
```

```
scambio i valori x = 7 y = 5  
passaggio per indirizzo : x = 5 y = 7
```

- ***overloading*** (*sovraccarico*)
- all'interno di uno stesso programma è possibile definire più funzioni aventi lo ***stesso nome*** purché sia ***differente*** la ***lista*** dei tipi dei ***parametri***
- es
  - `double max(double, double);`
  - `int max(int, int);`

- per i parametri *call-by-value* si può specificare un *valore di default*
- se il corrispondente argomento *manca*, il parametro assume il valore di default
- il valore di default va inserito nella prima tra dichiarazione e definizione
- i parametri con valore di default devono stare nelle posizioni *più a destra*
- nella *chiamata* gli argomenti vanno *omessi* a partire da *destra*

```
#include <iostream>
using namespace std;
void showVolume(int length, int width = 1, int height = 1);
//Returns the volume of a box.
//If no height is given, the height is assumed to be 1.
//If neither height nor width are given, both are assumed to be 1.

int main( ) {
    showVolume(4, 6, 2);
    showVolume(4, 6);
    showVolume(4);
    return 0;
}

void showVolume(int length, int width, int height) {
    cout << "Volume of a box with \n"
        << "Length = " << length << ", Width = " << width << endl
        << "and Height = " << height
        << " is " << length*width*height << endl;
}
```