FUNZIONI - C++

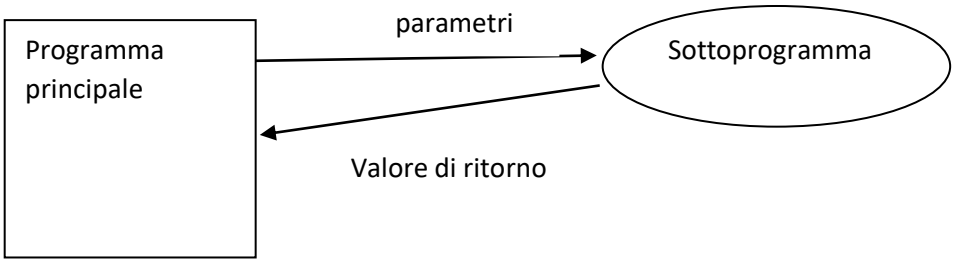
SIGNIFICATO

Una funzione è un sottoprogramma, ovvero una particolare sezione di codice che può venire mandata in esecuzione su richiesta del programma principale.

Una funzione viene realizzata per svolgere dei compiti specifici (ad esempio calcolare somma degli elementi di un array). Si dice che una funzione ha una responsabilità: per ogni funzione che creiamo dovremmo chiederci: *Cosa calcola questa funzione? Di cosa ha bisogno per farlo?*

Una funzione può venire invocata (chiamata) più volte dal programma principale.

Una funzione può ricevere dell'input dal programma principale e emettere dell' output che sarà utilizzato sempre nel programma principale.



- Il programma principale chiama il sottoprogramma passandogli dei parametri.
- Il sottoprogramma elabora.
- Il sottoprogramma restituisce il valore calcolato che può ora venire usato nel programma principale.

PARAMETRI E VALORE RESTITUITO

I parametri sono il modo in cui il programma principale da in input dei valori al sottoprogramma.

- Un parametro ha un tipo e un nome.
- Un parametro si comporta come una variabile all'interno della funzione ed è visibile solo all'interno della funzione stessa.
- Il valore iniziale del parametro è quello assegnato durante la chiamata.

Il valore restituito dal sottoprogramma è il modo con cui il sottoprogramma comunica al programma principale il risultato della sua computazione.

- Il valore restituito determina il tipo della funzione: se il valore restituito dalla funzione è un intero, il tipo della funzione sarà int.
- Le funzioni che non hanno un valore restituito sono di tipo void.

SINTASSI

```
tipoFunzione nomeFunzione ( lista parametri ){
      // corpo della funzione
      return valore_di_ritorno;
}
```

I parametri sono opzionali.

Se l'istruzione return è presente, allora tipoFunzione non deve essere void.

ESEMPIO 1 - UNA FUNZIONE SENZA INPUT E DI TIPO VOID

- Vediamo una funzione base.
- Di tipo void che quindi non restituisce niente.
- Non ha parametri.
- Quando invocata fa sempre la stessa cosa: stampa la stringa “ciao”.

```
#include <iostream>
using namespace std;

void stampaCiao(){
	cout << "ciao" << endl;
}

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

//output:
//ciao
//ciao
```

ESEMPIO 2 – FUNZIONI CON PARAMETRI E VALORI DI RITORNO

Vediamo ora delle funzioni più complicate. La funzione *minimo* restituisce il minimo valore dei parametri *n1* e *n2*. La funzione *raddoppia* restituisce il doppio del valore di input *n.*

```
#include <iostream>
using namespace std;

int minimo(int n1, int n2){
	if(n1 < n2){
		return n1;
	}else{
		return n2;
	}
}

int raddoppia(int n){
	int risultato;
	risultato = n*2;
	return risultato;
}

int main(){
	int a = 2; int b = 3;

	// 1) calcoliamo il doppio del valore minimo tra a e b
	int min = minimo(a,b);
	int risultato = raddoppia(min);
	cout << risultato << endl;

	// 2) calcoliamo il doppo del valore minimo tra a e b
	//    ...ma usando meno righe di codice
	cout << raddoppia(minimo(a,b)) << endl;

	// 3) calcoliamo il valore minimo tra b e il doppio di a
	cout << minimo(raddoppia(a), b) << endl;
}

//output:
//4
//4
//3
```

Notiamo che chiamando una funzione di tipo diverso da void, possiamo assegnare il valore restituito ad una variabile e riutilizzare in seguito per successivi calcoli tale valore.

Vediamo anche che possiamo annidare le chiamate di funzioni; se tali funzioni fossero di tipo void e anziché restituire il valore lo stampassero, ciò non sarebbe possibile.

ESEMPIO 3 - UNA FUNZIONE CHE HA COME PARAMETRO UN ARRAY

- In una funzione posso parare più parametri.
- Posso passare anche array.
- Se passo un array devo passare anche la sua lunghezza (altrimenti non avrei modo di conoscerla all'interno della funzione).
- Nella chiamata della funzione vengono passati un array di interi e un intero, se così non fosse, il compilatore genererebbe un errore.
- Il valore restituito dalla chiamata della funzione è un intero e pertanto può venire assegnato a delle variabili, nel nostro caso viene assegnato alla variabile *somma.*

```
#include <iostream>
using namespace std;

int calcolaSommaArray(int arr[], int lunghezza){
	int risultato = 0;
	for(int i = 0; i < lunghezza; i++){
		risultato = risultato + arr[i];
	}
	return risultato;
}

int main(){
	int ilMioArray [5] = {1,2,3,4,5};
	int somma = calcolaSommaArray(ilMioArray, 5);
	cout << somma << endl;

	int numeriFortunati[10] = {1,1,1,1,1,1,1,1,1,1};
	int altraSomma;
	altraSomma = calcolaSommaArray(numeriFortunati, 10);
	cout << altraSomma;

	return 0;
}

//output:
//15
//10

//notare che è stato possibile passare array di lunghezza differente
```

ESEMPIO 4 - LE VARIABILI VENGONO PASSATE PER VALORE

- La funzione **azzeraNumero(int x)**prende in input un intero:
  - Lo stampa.
  - Pone il valore di x uguale a 0.
  - Stampa x.
- Si noti che la variabile che viene azzerata è la varabile x che è visibile solo all'interno della funzione.
- La variabile y non subisce effetti. Infatti nella chiamata **azzeraNumero(y);** viene passata alla funzione soltanto il **valore** di y (cioè 2).

```
#include <iostream>
using namespace std;

void azzeraNumero(int x){ // x è una nuova variabile
                          // che prende il VALORE dell'espressione
                          // passata nella chiamata di funzione
   cout << x <<endl;
   x = 0;
   cout << x <<endl;
}


int main(){
   int y = 2;
   cout << y <<endl;
   azzeraNumero(y);   // Passaggio del VALORE di y
   cout << y <<endl;
}

//output:
//2
//2
//0
//2

//notare che y non è stata azzerata!
```

Come vedremo nell'esempio 8 è possibile modificare questo comportamento.

ESEMPIO 5 - GLI ARRAY VENGONO PASSATI PER RIFERIMENTO

- La funzione **stampaArray (int array[], int lunghezzaArray)**, scorre tutto l'array e stampa ciascun valore.
- La funzione **azzeraArray(int array[], int lunghezzaArray)** , scorre tutto l'array e assegna 0 ad ogni posizione dello stesso.
- La chiamata **azzeraArray(ilMioArray, 5);** provoca l'azzeramento di tutto l'array *ilMioArray* e i cambiamenti sono tangibili anche nel main.

```
#include <iostream>
using namespace std;

void stampaArray(int array[], int lunghezzaArray){
   for(int i = 0; i < lunghezzaArray; i++){
        cout << array[i] << " ";
   }
   cout <<endl;
}

void azzeraArray(int array[], int lunghezzaArray){
   for(int i = 0; i < lunghezzaArray; i++){
        array[i] = 0;
   }
}

int main(){
   int ilMioArray [5] = {1,2,73,34,45};

   stampaArray(ilMioArray, 5); // Array passato per RIFERIMENTO

   azzeraArray(ilMioArray, 5); // Array passato per RIFERIMENTO

   stampaArray(ilMioArray, 5); // Array passato per RIFERIMENTO

   return 0;
}

//output:
//1 2 73 34 45
//0 0 0 0 0
```

ESEMPIO 6 – LE STRINGHE VENGONO PASSATE PER VALORE

```
#include <iostream>
#include <string> // ci permette di usare le stringhe
#include <cctype> // aggiunge le funzioni tolower e toupper
using namespace std;

void convertiInMaiuscolo(string parola){
   for(int i = 0; i < parola.length(); i++){
      parola[i] = toupper(parola[i]);
   }
}

int main(){
   string nome = "mario";

   convertiInMaiuscolo(nome);
   cout << nome;

   return 0;
}

//output:
//mario
```

In questo esempio abbiamo una funzione che prende in input una parola e la converte in maiuscolo. Tuttavia, anche le stringhe sono passate per valore quindi nel main non vediamo gli effetti apportati dalla funzione.

ESEMPIO 7 – GLI ARRAY DI CHAR VENGONO PASSATI PER RIFERIMENTO

```
#include <iostream>
#include <string.h> // ci permette di usare gli array di char
#include <cctype>   // aggiunge le funzioni tolower e toupper
using namespace std;

void convertiInMaiuscolo(char parola[]){
   int l = strlen(parola);
   for(int i = 0; i < l; i++){
      parola[i] = toupper(parola[i]);
   }
}

int main(){
   char nome[6] = "mario";

   convertiInMaiuscolo(nome);
   cout << nome;

   return 0;
}

//output:
//MARIO
```

Esempio analogo al precedente ma in questo caso abbiamo utilizzato gli array di char al posto delle stringhe. Avendo utilizzato degli array, questi vengono passati per riferimento e quindi la funzione fa il suo dovere convertendo tutto in maiuscolo.

ESEMPIO 8 – PASSARE VARIABILI DI TIPO INT, CHAR, BOOL,… PER RIFERIMENTO

Nell'esempio 4 abbiamo visto che i numeri interi (e i tipi di base tra cui bool, float ecc…) vengono passati per valore per impostazione predefinita.
Per fare in modo che una variabile sia passata per riferimento basta scrivere '&' vicino al nome del parametro nella dichiarazione della funzione.

```
#include <iostream>
using namespace std;

void azzeraNumero(int &x){
    cout << x <<endl;
    x = 0;
    cout << x <<endl;
}


int main(){
    int y = 2;
    cout << y <<endl;
    azzeraNumero(y);    // Passaggio del RIFERIMENTO di y
    cout << y <<endl;
}

//output
//2
//2
//0
//0

//notare che y VIENE azzerata!
```

Passando quindi y per riferimento, i cambiamenti effettuati su di essa all'interno del main sono visibili.

ESEMPIO 9 – SWAP BY VALUE AND BY REFERENCE

La funzione *scambia* ha come obbiettivo prendere in input due variabili (ad esempio di tipo *int*) e scambiarne il loro valore.
Ovvero se x1 vale 3 e x2 vale 5, dopo aver richiamato la funzione scambia x1 deve valere 5 e x2 deve valere 3.
Proviamo due approcci ossia passando le variabili alla funzione *scambia* per riferimento e per valore e vediamo i risultati.

```
#include <iostream>
using namespace std;

void scambiaPerRiferimento(int &x1, int &x2){
    int temp = x1;
    x1 = x2;
    x2 = temp;
}
void scambiaPerValore(int x1, int x2){
    int temp = x1;
    x1 = x2;
    x2 = temp;
}
int main(){
    int x1 = 3, x2 = 5;
    cout << "x1:" << x1 <<", x2:" << x2 <<endl;

    scambiaPerRiferimento(x1, x2);
    cout << "x1:" << x1 <<", x2:" << x2 <<endl;

    scambiaPerValore(x1, x2);
    cout << "x1:" << x1 <<", x2:" << x2 <<endl;
}

//output
//x1:3, x2:5
//x1:5, x2:3
//x1:5, x2:3
```

Come potevamo aspettarci solo passando gli argomenti per riferimento otteniamo il risultato sperato. Infatti *scambiaPerRiferimento* scambia effettivamente i valori di x1 e x2 portandoli rispettivamente a 5 e 3, ma *scambiaPerValore* li lascia inalterati (quindi rispettivamente 5 e 3).

Per completezza propongo un terzo approccio valido e funzionante come *scambiaPerRiferimento* ossia utilizzando i puntatori.

```
void scambiaPerPuntatori(int *x1, int *x2){
    int temp = *x1;
    *x1 = *x2;
    *x2 = temp;
}
int main(){
    int x1 = 3, x2 = 5;
    cout << "x1:" << x1 <<", x2:" << x2 <<endl;
    scambiaPerPuntatori(&x1, &x2);
    cout << "x1:" << x1 <<", x2:" << x2 <<endl;
}
//output
//x1:3, x2:5
//x1:5, x2:3
```

PASSAGGIO PER RIFERIMENTO E PER VALORE: RIASSUNTO

- Le variabili semplici (bool, float, int, double, char, string, …) sono passate per impostazione predefinita alle funzioni per **valore.** Il valore della variabile viene copiato nel rispettivo parametro e modificando il parametro non modifichiamo la variabile originale. Il parametro e la variabile passata si trovano in **due zone** di memoria differenti.
- Gli array sono passati alle funzioni per **riferimento**. Ovvero nella funzione il parametro fa riferimento allo stesso array. In memoria esiste **un'unica zona** in cui l'array è memorizzato.
- Possiamo passare anche le variabili semplici (bool, float, int, double, char, string, …) per riferimento semplicemente scrivendo **&** prima del nome del parametro nella definizione della funzione. Anche in questo caso ci sarà dunque un'unica zona di memoria in cui la variabile viene memorizzata.

APPROCCIO: PROGRAMMARE CON LE FUNZIONI

1) Scomporre il problema: dato un problema difficile lo scompongo in tanti sottoproblemi "facili".
2) Creare quindi una funzione per ciascun sotto problema.
3) Con opportune chiamate a sottoprogrammi comporre il programma principale che risolve il problema di partenza.

ESERCIZI

1) Creare una funzione di tipo bool, che prenda in input un numero, e restituisca true se il numero è pari.
2) Creare una funzione che prenda in input un numero *n* e che stampi "ciao" n volte.
3) Creare una funzione che prenda in input un numero *n* e una stringa *str* e che stampi *str n* volte.
4) Creare una funzione che prenda in input un array di interi *arr*, la sua lunghezza *len* e un numero *x*. La funzione restituisce true se il valore contenuto in *x* compare almeno una volta all'interno dell' array.
5) Creare una funzione che prenda in input un array e lo riempa di numeri casuali.
6) Creare una funzione che prenda in input una stringa e la stampi al contrario.
7) Creare una funzione che prenda in input un intero *len*, un primo array di interi *arr1* di lunghezza *len* e un secondo array di interi *arr2* sempre di lunghezza *len,* e restituisca true se i 2 array contengono gli stessi elementi in ogni posizione.
8) Creare una funzione che prenda in input un carattere e restituisca un intero. Se viene passato 'a', viene restituito 1; se viene passato 'b', viene restituito 2; se viene passato 'c', viene restituito '3'…
9) Creare una funzione che prenda in input una stringa e un array di interi di lunghezza predefinita 100. Sfruttando la funzione dell'esercizio precedente, viene scritto in ogni posizione dell' array un numero corrispondente alla lettera della stringa in tale posizione. (Ad esempio se la stringa vale "ciao", nelle prime 4 posizioni dell' array compariranno i numeri 3,9,1,15).

Roberto Borelli – Febbraio 2022

APPROFONDIMENTI

[1] https://www.html.it/pag/15497/le-funzioni1/

[2] https://www.html.it/pag/62163/funzioni-concetti-avanzati/

[3] https://www.youtube.com/watch?v=F454xj2a9rc

[4] https://www.youtube.com/watch?v=xmyi-BgBnB4