# I sottoprogrammi

Sono una sequenza di istruzioni che operano su dei dati producendo eventualmente dei risultati. Possono essere riutilizzati più volte durante l'esecuzione di un programma e ogni volta operare su dati differenti. I dati che vengono trasmessi al sottoprogramma si chiamano parametri attuali. I dati su cui operano i sottoprogrammi si chiamano parametri formali.

I parametri possono essere "passati" al sottoprogramma per valore o per riferimento.

Quando il **passaggio dei parametri** avviene per valore, al sottoprogramma viene in effetti passata solo una copia dell'argomento. Nella dichiarazione del sottoprogramma si dovrà definire il tipo di dato (che deve essere lo stesso del parametro ricevuto) e un identificatore. Grazie a questo meccanismo il valore della variabile nel programma chiamante non viene modificato.

Quando il **passaggio di parametri** avviene per riferimento, al sottoprogramma viene passato l'indirizzo e non il valore dell'argomento. In questo caso nella dichiarazione del sottoprogramma davanti al nome del parametro si deve mettere il simbolo "&".

I sottoprogrammi si distinguono in Procedure e Funzioni.

Le procedure quando vengono richiamate possono ricevere 0, 1, 2 … parametri dal programma chiamante e restituire 0, 1, 2, … parametri. Una procedura si definisce con la seguente sintassi:

**void** Nome_Procedura (tipo_dato Par1, tipo_dato Par2….)

Esempio: Il programma richiama la procedura SOMMA che deve sommare due numeri, Num1 e Num2 e deve restituire il risultato. Nel programma main() si potrebbe avere:

```
main()
{ int Num1, Num2, Ris:
cin>>Num1;
cin>>Num2;
SOMMA(Num1, Num2, Ris) ;
cout<<Ris;}
```

Il sottoprogramma SOMMA si potrebbe scrivere nel modo seguente;

```
void SOMMA( int A, int B, int &C)
{ C=A+B;
return;}
```

Le funzioni restituiscono sempre almeno un valore e possono ricevere 0, 1, 2, … parametri.

Una funzione si definisce con la seguente sintassi:

**Tipo** Nome_Funzione (tipo_dato Par1, tipo_dato Par2, ….)

La stessa operazione precedente si potrebbe realizzare utilizzando una funzione. Il codice potrebbe essere il seguente:

```
main()
{ int Num1, Num2, Ris:
cin>>Num1;
cin>>Num2;
Ris=SOMMA1(Num1, Num2);
cout<<Ris;}
```

La funzione SOMMA1 potrebbe essere scritta così:

```
Int SOMMA1(Int A, Int B)
{ return (A+B) ;}
```

Come si può notare la funzione termina con l'istruzione return che assegna alla variabile Ris del programma principale il risultato dell'operazione A+B. Il tipo della funzione deve essere lo stesso della variabile che riceverà il risultato.

# Prototipi

Secondo lo standard ANSI (American National Standard Institute), tutti i sottoprogrammi devono avere un corrispondente prototipo. È buona norma far risiedere il prototipo in un file d'intestazione (file header, con estensione `.h`) anche se questa non è una regola obbligatoria. Un **prototipo di sottoprogramma** ha la seguente forma:

Procedura:

void Nome_Procedura (tipo Par1, tipo Par2, …);

Funzione:

tipo_restituito nome_funzione (tipo Par1, tipo Par2,...);

La scrittura della **funzione** vera e propria è essa stessa una porzione di codice C++ che normalmente segue il codice del programma `main()`. Una funzione può quindi assumere la seguente forma:

```
tipo_restituito nome_funzione(tipo e nome_argomenti)
{
  // dichiarazione dei dati e corpo della funzione
  return();
}
```

Si noti che la prima riga della funzione è identica al prototipo inserito nel file header, ma con un'importante differenza: manca il punto e virgola finale. Vediamo ora un esempio che mostra **un prototipo e una funzione** in un programma:

```
/*
 * Un semplice programma che illustra l'uso dei prototipi
 * di funzioni. La funzione, esegue la moltiplicazione tra
 * due interi e restituisce il risultato
 */

#include <iostream.h>

// Definizione del prototipo
int moltiplica(int x, int y);


main()
{
  int a = 5;
  int b = 6;
  int risultato;

  risultato = moltiplica(a,b);
  cout << Il risultato della moltiplicazione è: << risultato << endl;
}
// Dichiarazione della funzione
int moltiplica(int x, int y)
{
  int ris;
  ris = x * y;
  // Valore restituito dalla funzione
  return ris;
}
```