



Le funzioni

Verso la programmazione
organizzata



Funzione

Una funzione è un particolare **costrutto sintattico** di un determinato linguaggio di programmazione che permette di raggruppare, all'interno di un programma, una sequenza di istruzioni in un unico **blocco**, espletando così una specifica (e in generale più complessa) operazione sui dati del programma stesso in modo tale che, a partire da determinati input, restituisca determinati output.

I **vantaggi** di una funzione stanno nel fatto che può essere invocata in diversi punti del programma di cui fa parte ogni volta in cui si ha la necessità di farlo come se fosse una singola istruzione, senza la necessità di doverne riscrivere ogni volta il relativo codice, implementando dunque il cosiddetto **riuso** di codice, cui si aggiunge una più facile **manutenibilità** del codice all'interno del programma e una più facile **progettazione** del software secondo la classica filosofia del divide et impera.

Funzione

The diagram shows a C++ function definition: `int Sum(int x, int y) { return x + y; }`. Red arrows point from labels to specific parts of the code: 'TIPO DI DATO IN OUTPUT' points to 'int' at the start; 'NOME FUNZIONE' points to 'Sum'; 'PARAMETRO INPUT' points to 'int x' and 'int y'; 'OPERAZIONE DI COMPUTAZIONE' points to the '+' sign in the return statement; and 'RESTITUZIONE DEL RISULTATO' points to the 'return' keyword.

TIPO DI DATO IN OUTPUT

NOME FUNZIONE

PARAMETRO INPUT

```
int Sum(int x, int y)
{
    return x + y;
}
```

RESTITUZIONE DEL RISULTATO

OPERAZIONE DI COMPUTAZIONE

La funzione accetta i parametri in input **dichiarando** due variabili x e y.

Le variabili x e y, essendo state dichiarate all'interno della funzione, termineranno il loro ciclo di vita una volta terminata la funzione.

Con la parola chiave **return** la funzione conclude il suo compito.

La funzione ricalca il modello concettuale dell'**algoritmo** e della **blackbox**.

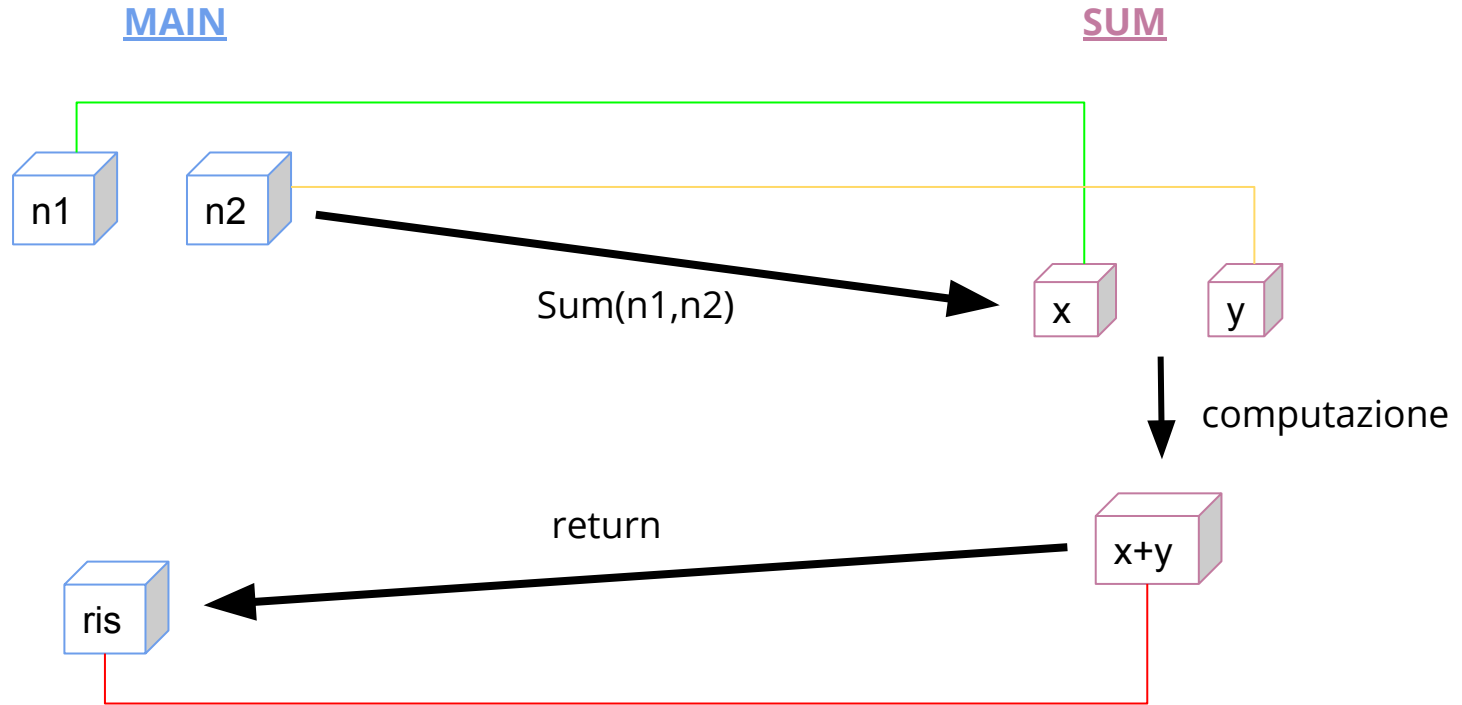
Invocazione di funzione

```
void Main()  
{  
    int n1 = 5, n2 = 7;  
    int ris = Sum(n1, n2);  
    Console.WriteLine(ris); // 12  
}
```

INVOCAZIONE DI
FUNZIONA

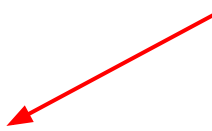


Passaggio di parametri per valore



Funzioni senza risultato

RISULTATO "VUOTO"



```
void PrintMyMessage(string msg)
{
    Console.WriteLine("Message: " + msg);
}
```

Sono funzioni che solitamente effettuano azioni, non calcoli.

Nelle funzioni void **MAI** mettere il return!

Funzioni senza input

```
int GetRandomNumber()  
{  
    Random r = new Random();  
    return r.Next();  
}
```

NESSUN INPUT



In questi casi l'operazione svolta dalla funzione non necessita di parametri di input, quindi il chiamante non li fornisce dato che non sono richiesti.

Output multiplo

Abbiamo visto che una funzione può ricevere in input nessuno, uno, due, tre, tanti parametri di input e restituire in output al più un valore.

Esiste un modo che permette a una funzione di restituire in output più di un valore?



Output con le struct

Quando una funzione necessita di produrre in output più valori o un risultato complesso, allora possiamo usare le struct come contenitori.

Immaginiamo di programmare una funzione che somma due numeri complessi forniti in input e produce in output il risultato.

```
struct Complex
{
    public int Re, Im;
}

Complex SumComplex(Complex c1, Complex c2)
{
    Complex ris = new Complex();
    ris.Re = c1.Re + c2.Re;
    ris.Im = c1.Im + c2.Im;
    return ris;
}
```

Output con le struct

Il programma principale prepara i dati di input e li passa alla funzione, la quale effettua il calcolo e lo restituisce al Main che lo memorizza nella variabile c3.

```
int Main()  
{  
    Complex c1 = new Complex();  
    c1.Re = 4;  
    c1.Im = 2;  
    Complex c2 = new Complex();  
    c2.Re = -1;  
    c2.Im = 8;  
  
    Complex c3 = SumComplex(c1, c2);  
    // .....  
}
```

Passaggio di parametri per riferimento

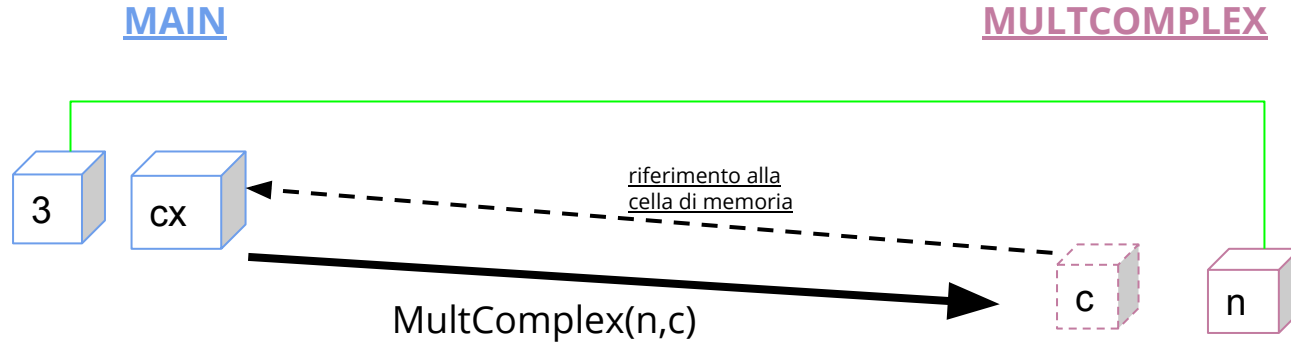
Ci sono delle situazioni in cui si desidera che la funzione modifichi gli stessi parametri di input invece che produrre un nuovo output come abbiamo visto nelle slide precedenti.

La parola chiave **ref** permette alla funzione di manipolare il dato originale, questo grazie alla tecnica del passaggio di parametro per riferimento.

```
void MultComplex(int n, ref Complex c)
{
    c.Re = c.Re * n;
    c.Im = c.Im * n;
}

int Main()
{
    Complex cx = new Complex();
    cx.Re = 4;
    cx.Im = 7;
    MultComplex(3, ref cx);
    Console.WriteLine(cx.Re); // 12
    Console.WriteLine(cx.Im); // 21
}
```

Passaggio di parametri per riferimento



Esercizi

Scrivere le seguenti funzioni:

1. stampa di un array
2. stampa di una matrice
3. controllo se una stringa è palindroma
4. calcolo dell'area di un trapezio
5. calcolo dell'elevamento a potenza
6. controllo di primalità di un numero
7. stampa di una struct di tipo Libro (vedi slide 8)

Esercizio Biblioteca

Facendo uso di struct, strutture dati e funzioni, creare un progetto console che permetta di gestire una biblioteca.

Il programma deve permettere di:

- memorizzare libri, tesserati e prestiti
- elencare i prestiti di un certo libro
- elencare i prestiti di un tesserato
- elencare i prestiti scaduti alla data attuale