

# Funzioni

Una funzione è un blocco di codice autonomo che esegue un'elaborazione

Sono fondamentali per due motivi:

- Suddividono il programma in parti più ridotte e lo rendono comprensibile
- Può essere richiamata, all'interno di un programma, tutte le volte che è necessario senza ripetere ogni volta le istruzioni in essa contenute

Cos'è una funzione?

```
tipo_restituito nome_funzione (elenco_parametri)
{
    corpo_della_funzione
}
```

## Forma generale

Definire una funzione significa creare e dotare di una nuova funzionalità il programma che si sta scrivendo

# Forma generale

## *tipo\_restituito*

- Specifica il tipo di dato restituito dalla funzione
- Una funzione può restituire un qualsiasi tipo di dato, tranne un array

## *elenco\_parametri*

- È un elenco di nomi di variabili separati da virgole e preceduti dai rispettivi tipi
- Tali variabili sono destinate a ricevere i valori degli argomenti nel momento in cui la funzione viene chiamata
- Una funzione può anche non richiedere parametri, ma l'uso delle parentesi è comunque obbligatorio

Abbiamo già  
vista una  
funzione?

La funzione main è la  
funzione principale ed è  
obbligatorio che sia definita  
in ogni applicazione

Costituisce il cosiddetto *entry point*  
del programma, cioè il punto in cui  
inizia l'esecuzione del programma  
una volta compilato del codice

# La funzione main()

La sua intestazione è quella di una funzione standard

- Restituisce il valore 0, quindi un int
- Ha un nome univoco: main
- Non ha parametri di ingresso: ()

# Esempio funzione

---

```
int somma(int a, int b)
{
    //fai la somma di due numeri
    int c = a + b;
    cout << "La somma dei numeri che mi hai passato è: " << c << endl;
    return c;
}
```

# Funzione che ritorna void

---

Quando una funzione non restituisce alcun valore, viene anche chiamata procedura e il valore restituito mancante è specificato mediante la parola chiave void



```
void saluto()  
{  
    //scrivi un messaggio di saluto  
    cout << "Salve salvini vicini" << endl;  
}
```

Esempio funzione

Ma dove inseriamo le funzioni?

---

Prima della  
funzione main()

# Esercizio

Provate a scrivere un programma che implementa tre funzioni:

- leggiLato(), in cui prende in input la misura del lato
- calcolaQuadr(), in cui calcola l'area di un quadrato
- scriviQuadr(), in cui si stampa a video il valore dell'area del quadrato

# Ambito delle variabili

## 01

Nei programmi precedenti, tutte le variabili venivano dichiarate all'interno della funzione `main()` e venivano distrutte con essa una volta terminato il flusso delle sue istruzioni

## 02

Abbiamo sempre usato quindi *variabili locali*, ossia variabili che sono dichiarate e utilizzate nel corpo della funzione stessa

## 03

Con l'inserimento di nuove funzioni, si rende necessario definire il campo d'azione (appunto ambito) di ciascuna variabile, per controllare in quali funzioni deve essere disponibile

---

Teniamo conto che abbiamo un programma con due funzioni, `main()` e `leggi()`

---

Se noi definiamo una variabile nella funzione `main()`, essa sarà disponibile solo in essa e non per la funzione `leggi()`

---

Al contrario, se noi definiamo una variabile nella funzione `leggi()`, essa sarà disponibile solo in essa e non per la funzione `main()`

---

Esempio ambito variabili

1

Ma se noi volessimo usare una variabile in tutte le funzioni del programma, come potremmo fare?



2

Creiamo delle *variabili globali*, le quali vengono definite all'interno del contenitore del codice, non più all'interno di una qualche funzione

# Ambito delle variabili

# Esercizio

---

Provate a questo punto a riscrivere il programma di prima, usando le variabili globali

# Parametri di una funzione

La maggior parte delle volte, una funzione ha bisogno di conoscere dei valori per poter eseguire il suo corpo



A questo vengono incontro i *parametri* (dentro alle parentesi tonde), come definito precedentemente

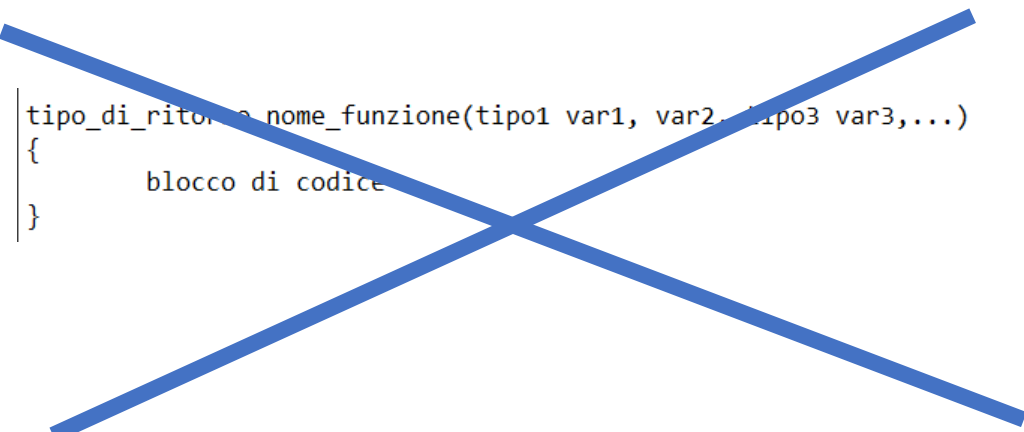
```
tipo_di_ritorno nome_funzione(tipo1 var1, tipo2 var2, tipo3 var3,...)
{
    blocco di codice
}
```



# Parametri di una funzione

---

Ogni parametro deve avere il suo valore corrispondente prima del suo nome, anche se ho più variabili con lo stesso tipo



```
tipo_di_ritorno nome_funzione(tipo1 var1, var2, tipo3 var3,...)
{
    blocco di codice
}
```

# Esempio utilizzo parametri

---

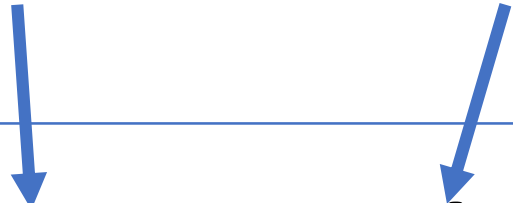
```
int somma(int a, int b){  
    int sum;  
    somma = a + b;  
    return sum;  
}  
  
int main(){  
    int num1;  
    int num2;  
    int numSomma;  
  
    cout<<"Dammi un numero"<<endl;  
    cin>>num1;  
    cout<<"Dammi un numero"<<endl;  
    cin>>num2;  
    numSomma = somma(num1, num2);  
}
```

# Passaggio per valore

---

```
c = somma(num1, num2)
```

---



```
int somma(int a, int b)
```

# Regole parametri della funzione

Gli argomenti passati come parametri alla funzione devono rispettare fedelmente sia il tipo di dato sia l'ordine di dichiarazione dell'intestazione della funzione

# Regole parametri della funzione

Nell'intestazione della funzione non è consentito assegnare valori ai parametri

# Regole parametri della funzione

Tutti i parametri passati per valore sono considerati locali alla funzione

# Regole parametri della funzione

L'invocazione di una funzione determina  
l'assegnazione degli argomenti ai relativi  
parametri

# Regole parametri della funzione

L'assegnazione in questione è definita *passaggio degli argomenti per valore*, in quanto a ogni parametro viene assegnato il corrispondente valore dell'argomento



## Regole parametri della funzione

In altre parole, per ogni argomento viene generata una copia che viene assegnata al corrispondente parametro, senza riflettersi sugli argomenti originariamente passati alla funzione

E se invece..

Volessimo modificare  
anche i valori che ci  
vengono passati  
come argomento?



Possiamo usare il  
passaggio per  
riferimento

## Passaggio per riferimento

---

Permette la modifica dei  
parametri in ingresso  
alla funzione

```
tipo_di_ritorno nome_funzione(tipo &var1, tipo &var2, ...){  
    //codice da eseguire  
}
```

# Passaggio per riferimento

---

Il carattere & che precede il parametro indica che ogni eventuale modifica effettuata su quel parametro si riflette sull'argomento dell'invocazione

# 01

Creare una funzione che incrementi il valore di due variabili intere passate alla funzione come parametri per valore, stampando il valore delle variabili sia prima di chiamare la funzione sia dopo la sua esecuzione

# 02

Creare una funzione che incrementi il valore di due variabili intere passate alla funzione come parametri per riferimento, stampando il valore delle variabili sia prima di chiamare la funzione sia dopo la sua esecuzione

## Esempio

# La libreria cmath

---

Per usare le funzioni matematiche più comuni, possiamo usare la libreria cmath, attraverso il comando `#include <cmath>`

---

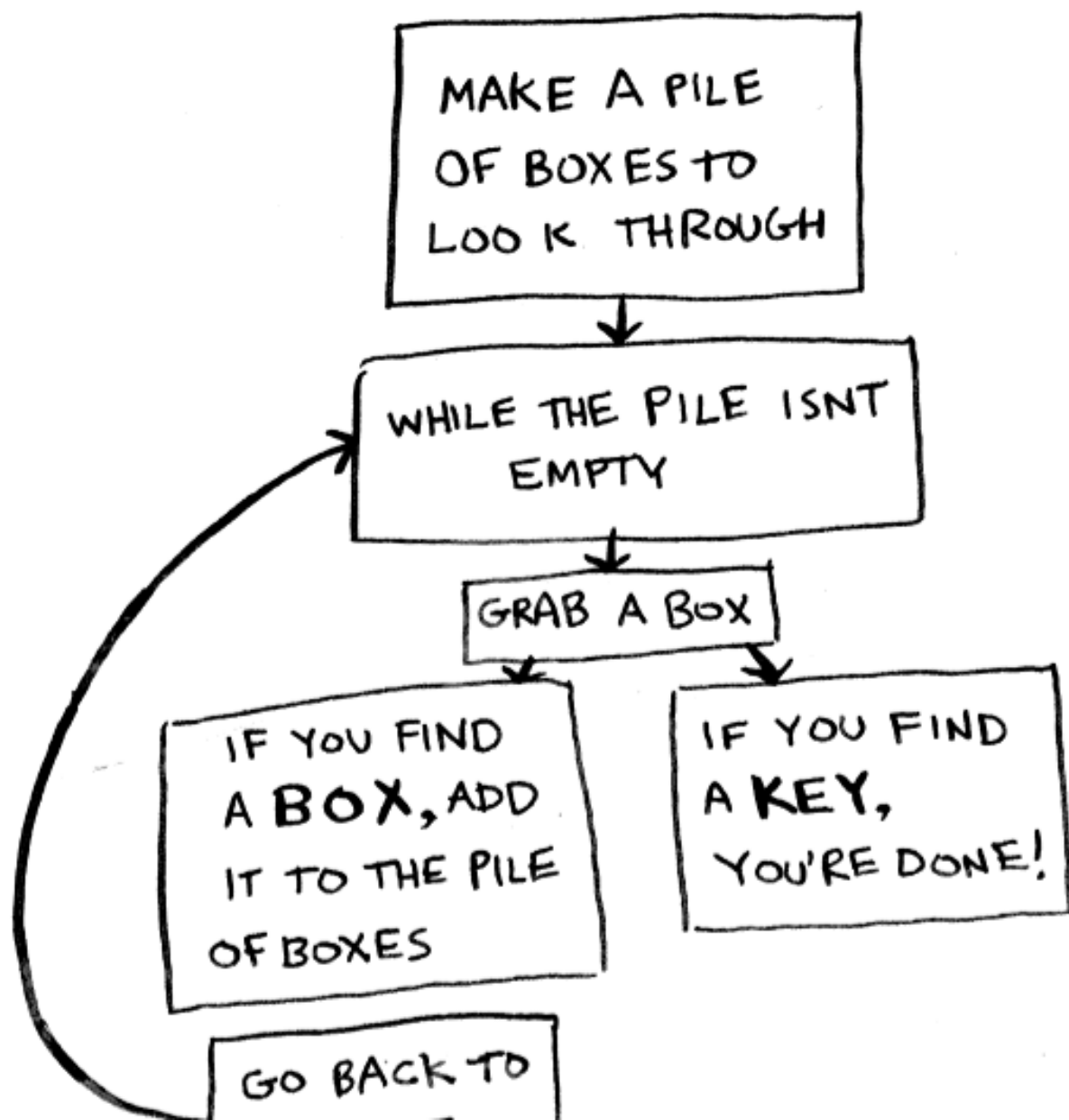
<http://www.cplusplus.com/reference/cmath/>



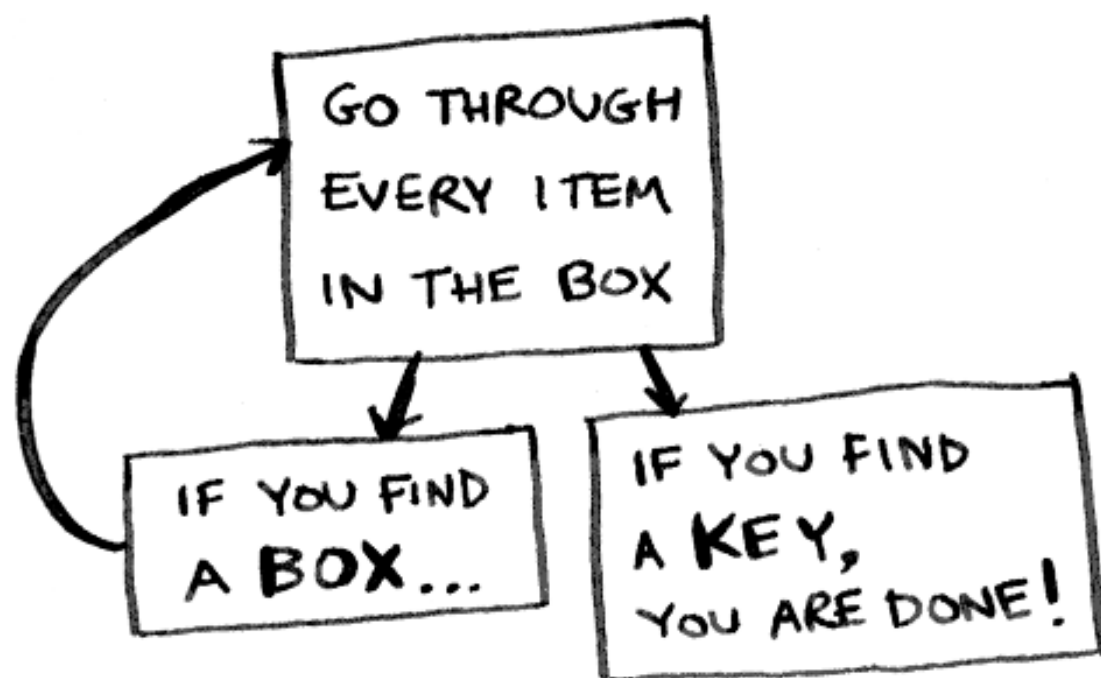


# Ricorsione

# Iterative Approach



# Recursive Approach





La funzione richiama sé stessa

L'importante è non richiamarla con gli stessi parametri di prima

Col vuol dire ricorsione?

Un esempio?

$$fattoriale(n) = n! = \begin{cases} 1 \\ n * fattoriale(n - 1) \end{cases}$$

Codificando..

```
int fattoriale(int n){  
    int fatt;  
    if(n==1){  
        return 1;  
    } else {  
        fatt = n*fattoriale(n-1);  
        return fatt;  
    }  
}
```

main()

rfat(n)

n 3

rfat(n-1)

2

f 6

rfat(n)

n 2

rfat(n-1)

1

f 2

rfat(n)

n 1

rfat(n-1)

1

f 1

rfat(n)

n 0

6

2

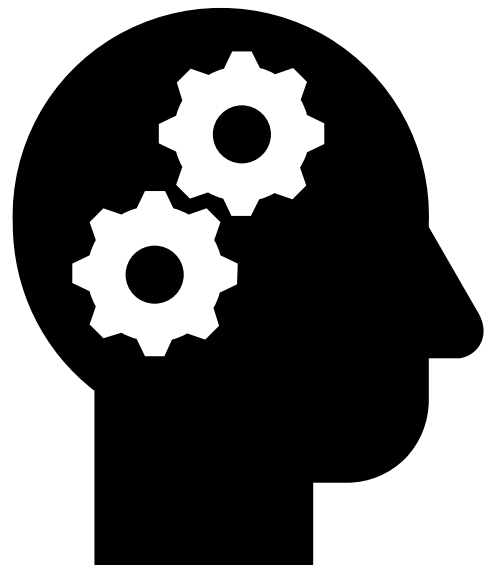
1

1

Entrata in  
loop

Dimenticanza  
dell'if di uscita

Errori comuni



Tempo di pratica