C++

```cpp
#include <iostream>

using namespace std;

int main() {
  cout << "ciao mondo!";
  return 0;
}
```

Include

```cpp
#include <iostream>
```

#include è una direttiva per il preprocessore o precompilatore che serve per includere file o librerie nel codice che si sta scrivendo. Includendo un file sarà possibile utilizzare tutte le classi, le funzioni, i namespace e le variabili presenti all'interno.

In questo caso si sta importando la libreria standard **iostream**, che serve per operazioni di input e output con la console.

Ci sono due tipi di #include:

- Con le parentesi angolari:

  ```cpp
  #include <iostream>
  ```

  Si utilizza per includere le librerie non presenti nella cartella in cui si sta lavorando ma inserite all'interno delle cartelle predefinite per la ricerca dei file del compilatore.

- Con le virgolette:

  ```cpp
  #include "file.cpp"
  ```

  Si utilizza per includere i file presenti nella cartella in cui si sta lavorando oppure nelle sottocartelle.

Namespace

Per capire cosa sono i namespace prima bisogna capire cos'è **lo scope**. In poche parole lo scope e uno spazio nel quale le variabili dichiarate possono essere utilizzate. In poche parole uno scope è definito da una coppia di parentesi graffe {}. Ad esempio:

```
int main() {
  // variabile
  int age = 0;

  // stampa la variabile
  cout << age;
  return 0;
}
```

Le variabili all'esterno di qualsiasi coppia di parentesi si chiamano **variabili globali**

```
// variabile
int age = 0;

int main() {
  // stampa la variabile
  cout << age;
  return 0;
}
```

Nello stesso scope non ci possono essere due variabili o funzioni con lo stesso nome e se se viene dichiarata una variabile con lo stesso nome in uno scope più basso sarà visibile solo quest'ultima variabile.

```
// variabile
int age = 0;

int main() {
  // variabile con lo stesso nome ma diverso valore
  int age = 57;

  // stampa la variabile
  cout << age; // 57
  return 0;
}
```

Uno scope può essere definito anche da una coppia di parentesi senza nome che non definisce una funzione o una classe.

```
int main() {
  // scope 1
  {
    int age = 57;
    cout << age;
  }

  // scope 2
  {
    int age = 20;
    cout << age;
  }

  // non funziona perché age non è nello scope ma in uno sottostante
  cout << age;
  return 0;
}
```

I name space sono stati introdotti per dare un nome a queste coppie vuote di parentesi e ai gruppi di variabili, classi e funzioni che hanno lo stesso scopo.

Tutta la libreria standard del c++ è stata inserita all'interno del namespace **std**.

Per utilizzare una funzione si utilizza la dicitura:

```
namespace::variabile
```

dove la :: si chiama **scope operator**.

Se si sta utilizzando molte volte lo stesso name space si può utilizzare una dicitura per dire al compilatore di cercare automaticamente le funzioni che non trova in un certo namespace:

```
using namespace std;
```

L'esempio del professore senza questa riga sarebbe così:

```cpp
#include <iostream>

int main() {
  std::cout << "Ciao mondo";
  return 0;
}
```

Funzioni

Un sottoprogramma o funzione è un insieme di istruzioni identificate da un nome che può essere richiamato all'interno del programma principale o all'interno dei sottoprogrammi.

```cpp
int main() {
  std::cout << "Ciao mondo";
  return 0;
}
```

Nel caso invece in cui la funzione restituisca un valore si specifica prima del nome della funzione stessa il tipo di dato che viene restituito. Se non si specifica nulla, allora in automatico si considera il valore di ritorno un intero. Se non si restituisce nessun valore si mette **void**.

Per restituire un valore si usa la parola **return** seguita dal valore che deve essere restituito. Ogni istruzione dopo il return viene ignorata.

Main è una funzione particolare perché è la prima che viene eseguita quando si avvia il programma.

Tra parentesi si mettono gli argomenti passati alla funzione, anche detti **parametri**.

```cpp
int somma(int a, int,b) {
  return a+b;
}

int main(){
  // chiamata della funzione
  int risultato = somma(50, 20);

  // verrà stampato 70
  cout << risultato;
}
```

Variabili

Le variabili si creano mettendo il tipo di dato, il nome della funzione, =, il valore e ;:

```
int nome = 5;
```

Se ad una variabile non viene assegnato nessun valore essa avrà il valore che è scritto nello spazio della RAM dove è fisicamente allocata la variabile.

Operatori

- *: moltiplicazione
- /: divisioni
- +: somma
- -: sottrazione
- %: modulo o resto della divisione tra interi
- += valore: incrementa la variabile del valore specificato

  ```
  variabile += 5;
  variabile = variabile + 5;
  ```

- -= valore: sottrae alla variabile il valore specificato

  ```
  variabile -= 5;
  variabile = variabile - 5;
  ```

Questi ultimi due operatori non aumentano la capacità espressiva del linguaggio e valgono ugualmente per la divisione e per la moltiplicazione.

- ++: incrementa di 1 il valore della variabile. Ci sono due modi per utilizzare questa operazione: se viene scritto variabile++, la variabile viene prima utilizzata nell'espressione e poi incrementata. Se si usa ++variabile è il contrario
- --: stessa cosa di ++ ma decrementa di 1;
- **bool**: può assumere solo due valori, vero o falso.

Tipi di dato

- **int**: numero intero con segno 32bit (-2.147.483.648, +2.147.483.647). il primo bit è il bit di segno
- **unsigned int**: numero intero senza segno 32bit (0, +4.294.967.295). il primo bit è il bit di segno
- **short int**: numero intero con segno 16bit (-32.768, +32.767). il primo bit è il bit di segno
- **short unsigned int**: numero intero senza segno 16 bit
- **float**: numeri in virgola mobile 32 bit
- **double**: numeri in virgola mobile 64 bit. Hanno più precisioni con numeri molto piccoli o molto grandi
- **char**: carattere singolo 8 bit in codice ASCII (ogni codice associato ad un simbolo) (-128, +127)
- **unsigned char**: char senza segno 8 bit (0, +255) nel codice le stringhe si indicano con le virgolette

```
char carattere = 'A';
```

I valori costanti si indicano con

```
const int variabile = 5;
```

Se un float viene moltiplicato con un int il risultato sarà un float, quindi il tipo più capiente. E' comunque preferibile fare operazioni con variabili dello stesso tipo per evitare conflitti ed errori.

Non posso assegnare un valore ad una variabile di un tipo diverso a parte per gli int: gli int sono contenuti nei float quindi posso assegnare un int ad un float, ma non un float ad un int.

Posso convertire un tipo ad un altro utilizzando l'operatore di **casting**, anche se a volte perdiamo alcune informazioni. Se questo operatore viene utilizzato durante una espressione viene cambiato solo momentaneamente il tipo alla variabile, non si modifica il valore o la variabile stessa. L'operatore di casting ha la precedenza sulle operazioni matematiche.

```
float a = 5.456;
int b = (int)a; // b = 5
```

La divisione tra numeri int risulterà un int, anche se ha il resto: 5/4 = 1. Tutte le altre risultano float.

Istruzioni di controllo

Servono per adattare il comportamento del programma a diverse situazioni. La prima è if.

```
if(condizione) {
    // operazioni in caso la condizione sia vera
} else {
    // operazioni in caso la condizione sia falsa
}
```

Se bisogna eseguire una sola operazione si possono omettere le parentesi graffe.

```
float a = 0;
float b = 1.345;
int c;

if(a != 0)
    c = b / a;
else
    c = a + b;
```

Ci sono diversi operatori per le condizioni:

- **==**: uguale
- **!=**: diverso

C	!(C)
V	F
F	V

- **> < >= <=**: maggiore, minore, maggiore o uguale, minore o uguale
- **||**: and

C1	C2	II
F	F	F
F	V	V
V	F	V
V	V	V

- **&&**: or

C1	C2	&&
F	F	F
F	V	F
V	F	F
V	V	V

Se un valore è 0 viene considerato falso, in qualsiasi altro caso viene considerato true.

Non si fanno confronti alla cazzo: solo due valori alla volta stronzo!!!!!!!!!!

Libreria standard

```
cout << risultato;
```

Cout è una proprietà della libreria standard che permette di stampare sulla console quello che vogliamo. Le due parentesi angolari sono un operatore <<.