C++

```cpp
#include <iostream>

using namespace std;

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

1. Include

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

#include è una direttiva per il preprocessore del compilatore 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.

2. 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:

```cpp
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**

```cpp
// 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.

```cpp
// 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.

```cpp
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:

```
#include <iostream>

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

3. 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.

```
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**.

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

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

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

4. 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 <<.