# **CPP**

|  |
| --- |
| // C++ program to display "Hello World"  // Header file for input output functions  #include <iostream>  **using** **namespace** std;    // Main() function: where the execution of program begins  **int** main()  {      // prints hello world      cout << "Hello World";    **return** 0;  } |

**Output**

Hello World

Let us now understand every line and the terminologies of the above program:

**1) // C++ program to display “Hello World”:** This line is a comment line. A comment is used to display additional information about the program. A comment does not contain any programming logic. When a comment is encountered by a compiler, the compiler simply skips that line of code. Any line beginning with ‘//’ without quotes OR in between /\*…\*/ in C++ is comment. Click to know [**More about Comments.**](https://www.geeksforgeeks.org/comments-in-c-c/)

**2) #include**: In C++,  all lines that start with pound (#) sign are called directives and are processed by a preprocessor which is a program invoked by the compiler. The **#include** directive tells the compiler to include a file and **#include<iostream>**. It tells the compiler to include the standard iostream file which contains declarations of all the standard input/output library functions. Click to Know [**More on Preprocessors.**](https://www.geeksforgeeks.org/cc-preprocessors/)

**3) using namespace std**: This is used to import the entirety of the std namespace into the current namespace of the program. The statement using namespace std is generally considered a bad practice. When we import a namespace we are essentially pulling all type definitions into the current scope. The std namespace is huge. The alternative to this statement is to specify the namespace to which the identifier belongs using the scope operator(::) each time we declare a type. Click to know [**More about using namespace std.**](https://www.geeksforgeeks.org/using-namespace-std-considered-bad-practice/)

**4) int main()**: This line is used to declare a function named “main” which returns data of integer type. A function is a group of statements that are designed to perform a specific task. Execution of every C++ program begins with the main() function, no matter where the function is located in the program. So, every C++ program must have a main() function. Click to know [**More about the main() function.**](https://www.geeksforgeeks.org/executing-main-in-c-behind-the-scene/)

**5) { and }**: The opening braces ‘{‘ indicates the beginning of the main function and the closing braces ‘}’ indicates the ending of the main function. Everything between these two comprises the body of the main function.

**6) std::cout<<“Hello World”;**:  This line tells the compiler to display the message “Hello World” on the screen. A semi-colon ‘;’ is used to end a statement. Semi-colon character at the end of the statement is used to indicate that the statement is ending there.

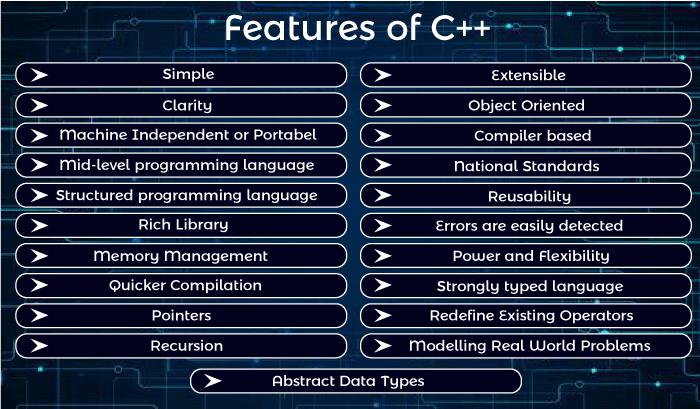
The std::cout is used to identify the standard character output device which is usually the desktop screen. Everything followed by the character “<<” is displayed to the output device. Click to know [**More on Input/Output.**](https://www.geeksforgeeks.org/basic-input-output-c/)

**7) return 0;** : This is also a statement. This statement is used to return a value from a function and indicates the finishing of a function. This statement is basically used in functions to return the results of the operations performed by a function.

**8) Indentation**: As you can see the cout and the return statement have been indented or moved to the right side. This is done to make the code more readable. In a program as Hello World, it does not hold much relevance, but as the programs become more complex, it makes the code more readable, less error-prone. Therefore, you must always use indentations and comments to make the code more readable. Must read the [**FAQ on the style of writing programs.**](https://www.geeksforgeeks.org/facts-and-question-related-to-style-of-writing-programs-in-c-c/)

**Important Points to Note while Writing a C++ Program:**

1. Always include the necessary header files for the smooth execution of functions.For example, **<iostream>** must be included to use **std::cin** and **std::cout**.
2. The execution of code begins from the **main()** function.
3. It is a good practice to use **Indentation** and **comments** in programs for easy understanding.
4. **cout** is used to print statements and **cin** is used to take inputs.



**#include<iostream.h>** includes the **standard input output** library functions. It provides **cin** and **cout** methods for reading from input and writing to output respectively.

**#include <conio.h>** includes the **console input output** library functions. The getch() function is defined in conio.h file.

|  |  |
| --- | --- |
| **Header File** | **Function and Description** |
| <iostream> | It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively. |
| <iomanip> | It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.** |
| <fstream> | It is used to declare services for user-controlled file processing. |

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**std::cout << "Hello world" << std::endl;**

**}**

Table

Description automatically generated

Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the datatype with which it is declared.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size (in bytes)** | **Range** |
| short int | 2 | -32,768 to 32,767 |
| unsigned short int | 2 | 0 to 65,535 |
| unsigned int | 4 | 0 to 4,294,967,295 |
| int | 4 | -2,147,483,648 to 2,147,483,647 |
| long int | 4 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 4 | 0 to 4,294,967,295 |
| long long int | 8 | -(2^63) to (2^63)-1 |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 |
| signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| float | 4 |  |
| double | 8 |  |
| long double | 12 |  |
| wchar\_t | 2 or 4 | 1 wide character |

**Note**: Above values may vary from compiler to compiler. In the above example, we have considered GCC 32 bit.

Graphical user interface, text, application

Description automatically generated

Deference b/w class and method

**C++ Functions**

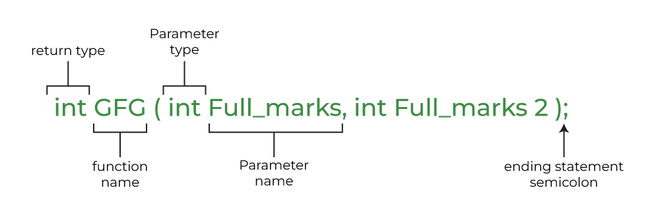
A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once and use it many times.

**Create a Function**

C++ provides some **pre-defined functions**, such as main(), which is used to execute code. But you can also create your own functions to perform certain actions.



// Create a function  
void myFunction() { //function Declaration part  
  cout << "I just got executed!";  
}  
int main() {  
  **myFunction();** // call the function  
  return 0;  
}  
// Outputs "I just got executed!"

**Note:** If a user-defined function, such as myFunction() is declared after the main() function, **an error will occur**:

Method:

for swap two element

**swap**(a[0], b[0])

# **Array:**

Array problem list: <https://www.geeksforgeeks.org/c-programs-gq/array-programs-gq/>

## **OOPs (Object Oriented Programming System)**



**Object:**

* Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.
* An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.
* Object take up space in memory and have an associated address like a record in pascal or structure or union in C.
* When a program is executed, the objects interact by sending messages to one another.
* Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.
* Object take up space in memory and have an associated address like a record in pascal or structure or union in C.
* When a program is executed the objects interact by sending messages to one another.
* Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

**Class:**

* Collection of objects is called class. It is a logical entity.
* A Class is a user-defined data-type which has data members and member functions.
* Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behavior of the objects in a Class.

class MyClass {       // The class  
  public:             // Access specifier  
    int myNum;        // Attribute (int variable)  
    string myString;  // Attribute (string variable)  
};  
int main() {  
  MyClass **myObj**;  // Create an object of MyClass  
  
  // Access attributes and set values  
  **myObj.myNum** = 15;   
  **myObj.myString** = "Some text";  
  
  // Print attribute values  
  cout << myObj.myNum << "\n";  
  cout << myObj.myString;  
  return 0;  
}

| **S. No.** | **Class** | **Object** |
| --- | --- | --- |
| 1 | Class is used as a template for declaring and  creating the objects. | An object is an instance of a class. |
| 2 | When a class is created, no memory is allocated. | Objects are allocated memory space whenever they are created. |
| 3 | The class has to be declared only once. | An object is created many times as per requirement. |
| 4 | A class cannot be manipulated as they are not available in the memory. | Objects can be manipulated. |
| 5 | A class is a logical entity. | An object is a physical entity. |
| 6 | It is declared with the class keyword | It is created with a class name in C++ and  with the new keywords in Java. |
| 7 | Class does not contain any values which  can be associated with the field. | Each object has its own values, which are associated with it. |
| 8 | A class is used to bind data as well as methods together as a single unit. | Objects are like a variable of the class. |
| 9 | Syntax for Declaring Class in C++:  class <classname> {  }; | **Syntax for Instantiating an object for a Class in C++:**  class Student {     public:        void put(){            cout<<“Function Called”<<endl;        }  };   // The class is declared here  int main(){           Student s1;   // Object created           s1.put();  } |
| 10 | Example: Bike | Example: Ducati, Suzuki, Kawasaki |

**Constructors** are special class members which are called by the compiler every time an object of that class is instantiated. Constructors have the same name as the class and may be defined inside or outside the class definition.  
There are 3 types of constructors:

* [Default constructors](https://www.geeksforgeeks.org/constructors-c/)
* Parameterized constructors
* [Copy constructors](https://www.geeksforgeeks.org/copy-constructor-in-cpp/)

// C++ program to demonstrate constructors

#include <bits/stdc++.h>

using namespace std;

class Geeks

{

public:

int id;

//Default Constructor

Geeks()

{

cout << "Default Constructor called" << endl;

id=-1;

}

//Parameterized Constructor

Geeks(int x)

{

cout << "Parameterized Constructor called" << endl;

id=x;

}

};

int main() {

// obj1 will call Default Constructor

Geeks obj1;

cout << "Geek id is: " <<obj1.id << endl;

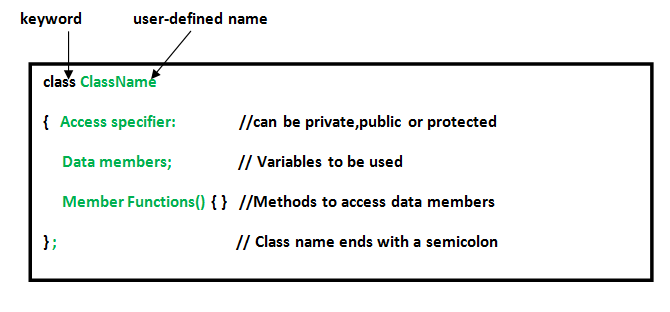
// obj2 will call Parameterized Constructor

Geeks obj2(21);

cout << "Geek id is: " <<obj2.id << endl;

return 0;

}



……………………………………………………………………………………………

**Inheritance**

inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”.

**Derived Classes:** A Derived class is defined as the class derived from the base class.  
**Syntax**:

class <derived\_class\_name> : <access-specifier> <base\_class\_name>

{

//body

}

**Where**  
**class**    — keyword to create a new class  
derived\_class\_name   — name of the new class, which will inherit the base class  
**access-specifier** — either of private, public, or protected. If neither is specified, PRIVATE is taken as default  
**base-**class-name — name of the base class

**Note**: A derived class doesn’t inherit ***access*** to private data members However, it does inherit a full parent object, which contains any private members which that class declares.

**Modes of Inheritance:**There are 3 modes of inheritance.

1. **Public Mode**: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.
2. **Protected Mode**: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.
3. **Private Mode**: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

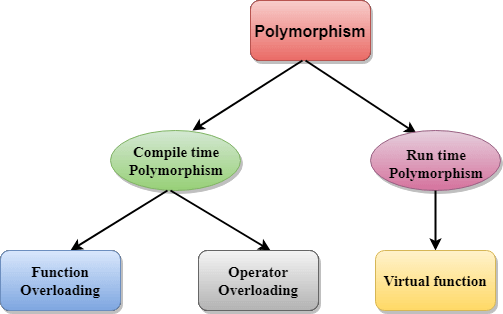
**Note:**The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed. For example, Classes B, C, and D all contain the variables x, y, and z in the below example. It is just a question of access.

**Types Of Inheritance: -**

1. Single inheritance: a class is allowed to inherit from only one class.
2. Multilevel inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class.
3. Multiple inheritance: a derived class is created from another derived class.
4. Hierarchical inheritance: more than one subclass is inherited from a single base class.
5. Hybrid inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance.

**Polymorphism**

**Polymorphism** is an important concept of object-oriented programming. It simply means more than one form. That is, the same entity (function or operator) behaves differently in different scenarios.



**Encapsulation**

**Encapsulation**is defined as wrapping up of data and information under a single unit.

* We cannot access any function from class directly. We need an object to access that function which is using the member the variable of that class.
* The function which we are making inside the class, it must use the all-member variable then only it is called encapsulation.

Abstraction is hiding the irrelevant information and providing only the essential information of the data to the outside world(user).

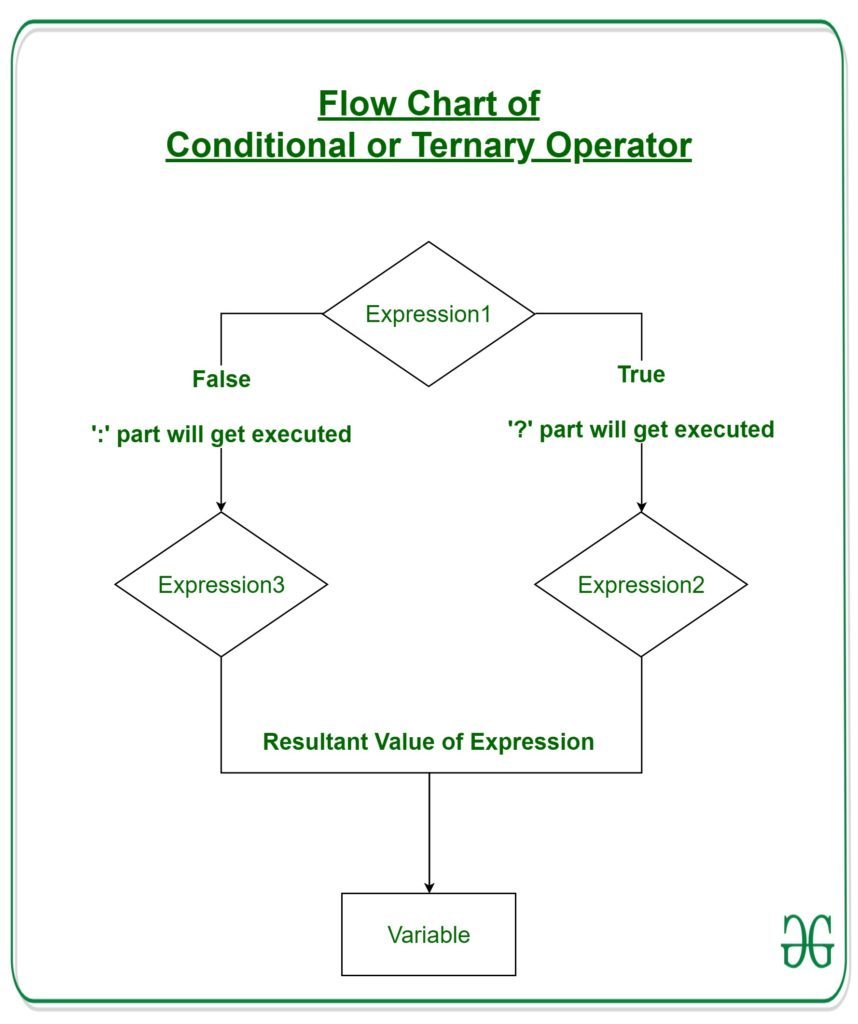
**public:** These data members are accessible both inside and outside the class.  
**private:** Private data members are not accessible outside the class and are not inheritable. They are available only inside the body of the class.  
**protected:** Protected members are not accessible outside the class but are inheritable in nature i.e. they are accessible to the derived/base class.

## **Getter (Accessor Function)**

## **Setter (Mutator Function)**

**\*\*Conditional or Ternary Operator (?:) in C/C++**

The conditional operator is kind of similar to the if-else statement as it does follow the same algorithm as of if-else statement, but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.



Text

Description automatically generated

# **Time Complexity**

**Time Complexity:** The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.

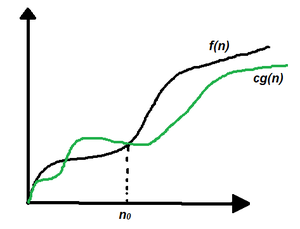
In detail: [Time Complexity Examples - Simplified 10 Min Guide (crio.do)](https://www.crio.do/blog/time-complexity-explained/)

# Big – Ω (Big- Omega) Notation

**Mathematical Representation:**

*Ω(g) = {f(n): there exist positive constants c and n0 such that 0 ≤ c\*g(n) ≤ f(n) for all n ≥ n0}   
Note: Ω (g) is a set*

**Graphical Representation:**



*Graphical Representation*

Follow the steps below to calculate Big – Omega(Ω) for any program:

1. Break the program into smaller segments.
2. Find the number of operations performed for each segment (in terms of the input size) assuming the given input is such that the program takes the least amount of time.
3. Add up all the operations and simplify it, let’s say it is f(n).
4. Remove all the constants and choose the term having the least order or any other function which is always less than f(n) when n tends to infinity, let say it is g(n) then, Big – Omega (Ω) of f(n) is Ω(g(n)).

Some examples:

Liner time = O(n)

Logarithm time > O(Log n )

Quadratic time >> O(n2)

Cubic time >> O(n3)

**Increasing complexity**

Min complexity O(1) >O(log n) >O(n)>> O(nlog n) >>O(n2) >>O(n3) >> O(2n) > O(n!) Max complexity