C++ Basic Tutorial

28 January 2022 16:40

© RAJAT KUMAR
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Following : https://www.w3schools.com/cpp/
C++ Core Guidelines :
GitHub - isocpp/CppCoreGuidelines: The C++ Core Guidelines are a set of tried-and-true guidelines, rules, and best practices about coding in C++

C++ Tutorial

Simple Program :

```
#include <iostream>    // #include<bits/stdc++.h>
using namespace std;

int main() {
  cout << "Hello World!";
  return 0;
}
```

Line 1: #include <iostream> is a header file library that lets us work with input and output objects, such as cout (used in line 5). Header files add functionality to C++ programs.

Line 2: using namespace std means that we can use names for objects and variables from the standard library.

Omitting Namespace

```
#include <iostream>

int main() {
  std::cout << "Hello World!";
  return 0;
}
```

Both \n and endl are used to break lines. However, \n is used more often and is the preferred way.

C++ Variables
Variables are containers for storing data values.

int - stores integers (whole numbers), without decimals, such as 123 or -123
double - stores floating point numbers, with decimals, such as 19.99 or -19.99
char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
string - stores text, such as "Hello World". String values are surrounded by double quotes
bool - stores values with two states: true or false

```
int myNum = 5;              // Integer (whole number without decimals)
double myFloatNum = 5.99;   // Floating point number (with decimals)
char myLetter = 'D';        // Character
string myText = "Hello";    // String (text)
bool myBoolean = true;      // Boolean (true or false)

int myAge = 35;
cout << "I am " << myAge << " years old.";

int x = 5, y = 6, z = 50;
cout << x + y + z;
```

C++ Identifiers
All C++ variables must be identified with unique names.

These unique names are called identifiers.

C++ Constants
When you do not want others (or yourself) to override existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only):

```
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

C++ User Input
```
int x;
cout << "Type a number: "; // Type a number and press enter
cin >> x; // Get user input from the keyboard
cout << "Your number is: " << x; // Display the input value
```

C++ Data Types

int	4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 7 decimal digits
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values

Scientific Numbers : A floating point number can also be a scientific number with an "e" to indicate the power of 10:
```
float f1 = 35e3;
double d1 = 12E4;
```

```
string greeting = "Hello";
cout << greeting;
```

C++ Operators
C++ divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

C++ Strings
```
// Include the string library
#include <string>

// Create a string variable
string greeting = "Hello";

string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;    // String Concatenation
string fullName = firstName.append(lastName);    // Append

string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is: " << txt.length();    // string length
```

txt.length() is same as txt.size();

cin considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only display a single word (even if you type many words):
```
string fullName;
cout << "Type your full name: ";
cin >> fullName;
cout << "Your name is: " << fullName;

// Type your full name: John Doe
```

```
// Your name is: John
```

when working with strings, we often use the getline() function to read a line of text. It takes cin as the first parameter, and the string variable as second:

```
string fullName;
cout << "Type your full name: ";
getline (cin, fullName);
cout << "Your name is: " << fullName;

// Type your full name: John Doe
// Your name is: John Doe
```

Omitting Namespace

You might see some C++ programs that runs without the standard namespace library. The using namespace std line can be omitted and replaced with the std keyword, followed by the :: operator for string (and cout) objects:

```
std::string greeting = "Hello";
std::cout << greeting;
```

Max and min

The max(x,y) function can be used to find the highest value of x and y:

And the min(x,y) function can be used to find the lowest value of x and y:

```
// Include the cmath library
#include <cmath>

cout << sqrt(64);
cout << round(2.6);
cout << log(2);
```

C++ If ... Else

```
int time = 22;
if (time < 10) {
  cout << "Good morning.";
} else if (time < 20) {
  cout << "Good day.";
} else {
  cout << "Good evening.";
}
// Outputs "Good evening."
```

Short Hand If...Else (Ternary Operator)

variable **= (***condition***) ?** *expressionTrue* **:** *expressionFalse***;**

```
int time = 20;
string result = (time < 18) ? "Good day." : "Good evening.";
cout << result;
```

C++ Switch Statements

```
int day = 4;
switch (day) {
  case 1:
    cout << "Monday";
    break;
  case 2:
    cout << "Tuesday";
    break;
  case 3:
    cout << "Wednesday";
    break;
  case 4:
```

```
    cout << "Thursday";
    break;
  case 5:
    cout << "Friday";
    break;
  case 6:
    cout << "Saturday";
    break;
  case 7:
    cout << "Sunday";
    break;
}
// Outputs "Thursday" (day 4)
```

The break Keyword

When C++ reaches a break keyword, it breaks out of the switch block.
This will stop the execution of more code and case testing inside the block.
When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default Keyword

The default keyword specifies some code to run if there is no case match:

The default keyword must be used as the last statement in the switch, and it does not need a break.

C++ While Loop

```
int i = 0;
while (i < 5) {
  cout << i << "\n";
  i++;
}
```

C++ Do/While Loop

```
int i = 0;
do {
  cout << i << "\n";
  i++;
}
while (i < 5);
```

C++ For Loop

```
for (int i = 0; i < 5; i++) {
  cout << i << "\n";
}

for (int i = 0; i <= 10; i = i + 2) {
  cout << i << "\n";
}
```

C++ Break

The break statement can also be used to jump out of a loop.

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  cout << i << "\n";
}
```

C++ Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

C++ Arrays

```
string cars[4];
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
int myNum[3] = {10, 20, 30};

string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cout << cars[0];
// Outputs Volvo

cars[0] = "Opel";   // Change an Array Element

string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
for(int i = 0; i < 4; i++) {
  cout << cars[i] << "\n";
}
```

Omit Array Size

You don't have to specify the size of the array. But if you don't, it will only be as big as the elements that are inserted into it:

```
string cars[] = {"Volvo", "BMW", "Ford"}; // size of array is always 3


string cars[5] = {"Volvo", "BMW", "Ford"}; // size of array is 5, even though it's only three elements inside it
cars[3] = "Mazda";
cars[4] = "Tesla";

string cars[5];
cars[0] = "Volvo";
cars[1] = "BMW";
...
```

C++ References

A reference variable is a "reference" to an existing variable, and it is created with the & operator:

```
string food = "Pizza";    // food variable
string &meal = food;   // reference to food

cout << food << "\n";  // Outputs Pizza
cout << meal << "\n";  // Outputs Pizza
```

Both food and meal have same object, is we change food or meal other one will automatically change.

C++ Memory Address

The & operator was used to create a reference variable. But it can also be used to get the memory address of a variable; which is the location of where the variable is stored on the computer.

```
string food = "Pizza";
cout << &food; // Outputs 0x6dfed4
```

The memory address is in hexadecimal form (0x..).

C++ Pointers

A pointer however, is a variable that stores the memory address as its value.

```
string food = "Pizza";  // A food variable of type string
string* ptr = &food;    // A pointer variable, with the name ptr, that stores the address of food

// Output the value of food (Pizza)
cout << food << "\n";

// Output the memory address of food (0x6dfed4)
cout << &food << "\n";

// Output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";

string* mystring; // Preferred
string *mystring;
string * mystring;
```

C++ Dereference

```
string food = "Pizza";  // Variable declaration
string* ptr = &food;    // Pointer declaration

// Reference: Output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";

// Dereference: Output the value of food with the pointer (Pizza)
cout << *ptr << "\n";
```

Note that the * sign can be confusing here, as it does two different things in our code:

- When used in declaration (string* ptr), it creates a pointer variable.
- When not used in declaration, it act as a dereference operator.

C++ Modify Pointers

```
string food = "Pizza";
string* ptr = &food;

// Output the value of food (Pizza)
cout << food << "\n";

// Output the memory address of food (0x6dfed4)
cout << &food << "\n";

// Access the memory address of food and output its value (Pizza)
cout << *ptr << "\n";

// Change the value of the pointer
*ptr = "Hamburger";

// Output the new value of the pointer (Hamburger)
cout << *ptr << "\n";

// Output the new value of the food variable (Hamburger)
cout << food << "\n";
```

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.

```
void myFunction(string country = "Norway") {
  cout << country << "\n";
}
```

```
int main() {
  myFunction("Sweden");
  myFunction("India");
  myFunction();
  myFunction("USA");
  return 0;
}

// Sweden
// India
// Norway
// USA
```

Pass By Reference

```
void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}

int main() {
  int firstNum = 10;
  int secondNum = 20;

  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  // Call the function, which will change the values of firstNum and secondNum
  swapNums(firstNum, secondNum);

  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  return 0;
}
```

Pas By Pointers

```
#include <iostream>
using namespace std;

void swap(int *x, int *y)
{
   int z = *x;
   *x = *y;
   *y = z;
}

// Driver Code
int main()
{
   int a = 45, b = 35;
   cout << "Before Swap\n";
   cout << "a = " << a << " b = " << b << "\n";

   swap(&a, &b);

   cout << "After Swap with pass by pointer\n";
   cout << "a = " << a << " b = " << b << "\n";
}
```

C++ Function Overloading

With function overloading, multiple functions can have the same name with different parameters:

```
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

C++ OOP
OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

What are Classes and Objects?
Classes and objects are the two main aspects of object-oriented programming.

Class : Fruit
Objects : Apple, Banana, Mango

a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and functions from the class.

C++ Classes/Objects
Everything in C++ is associated with classes and objects, along with its attributes and methods.
In real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

Attributes and methods are basically variables and functions that belongs to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

Create 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;
}
```

Class Methods
Methods are functions that belongs to the class.

There are two ways to define functions that belongs to a class:
- Inside class definition
- Outside class definition

Inside Example
```
class MyClass {      // The class
  public:          // Access specifier
    void myMethod() {  // Method/function defined inside the class
      cout << "Hello World!";
    }
```

```
};

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

Outside Example

```
class MyClass {       // The class
  public:             // Access specifier
    void myMethod();  // Method/function declaration
};

// Method/function definition outside the class
void MyClass::myMethod() {
  cout << "Hello World!";
}

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

C++ Constructors

A constructor in C++ is a special method that is automatically called when an object of a class is created.

```
class MyClass {     // The class
  public:           // Access specifier
    MyClass() {     // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;   // Create an object of MyClass (this will call the constructor)
  return 0;
}

class Car {       // The class
  public:         // Access specifier
    string brand;  // Attribute
    string model;  // Attribute
    int year;      // Attribute
    Car(string x, string y, int z) { // Constructor with parameters
      brand = x;
      model = y;
      year = z;
    }
};

int main() {
  // Create Car objects and call the constructor with different values
  Car carObj1("BMW", "X5", 1999);
  Car carObj2("Ford", "Mustang", 1969);

  // Print values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```

Access Specifiers

In C++, there are three access specifiers:

- public - members are accessible from outside the class
- private - members cannot be accessed (or viewed) from outside the class
- protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

C++ Encapsulation

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class). If you want others to read or modify the value of a private member, you can provide public get and set methods. (Using getters and setters)

```
class Employee {
  private:
    // Private attribute
    int salary;

  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};

int main() {
  Employee myObj;
  myObj.setSalary(50000);
  cout << myObj.getSalary();
  return 0;
}
```

C++ Inheritance

In C++, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- derived class (child) - the class that inherits from another class
- base class (parent) - the class being inherited from

To inherit from a class, use the : symbol.

```
// Base class
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n" ;
```

```
  }
};

// Derived class
class Car: public Vehicle {
 public:
  string model = "Mustang";
};

int main() {
 Car myCar;
 myCar.honk();
 cout << myCar.brand + " " + myCar.model;
 return 0;
}
```

Multilevel Inheritance
A class can also be derived from one class, which is already derived from another class.

```
// Base class (parent)
class MyClass {
 public:
  void myFunction() {
   cout << "Some content in parent class." ;
  }
};

// Derived class (child)
class MyChild: public MyClass {
};

// Derived class (grandchild)
class MyGrandChild: public MyChild {
};

int main() {
 MyGrandChild myObj;
 myObj.myFunction();
 return 0;
}
```

Multiple Inheritance
A class can also be derived from more than one base class, using a comma-separated list:

```
// Base class
class MyClass {
 public:
  void myFunction() {
   cout << "Some content in parent class." ;
  }
};

// Another base class
class MyOtherClass {
 public:
  void myOtherFunction() {
   cout << "Some content in another class." ;
  }
};

// Derived class
```

```
class MyChildClass: public MyClass, public MyOtherClass {
};

int main() {
  MyChildClass myObj;
  myObj.myFunction();
  myObj.myOtherFunction();
  return 0;
}
```

Access Specifiers

we have only used public (members of a class are accessible from outside the class) and private (members can only be accessed within the class). The third specifier, protected, is similar to private, but it can also be accessed in the inherited class:

```
// Base class
class Employee {
  protected: // Protected access specifier
    int salary;
};

// Derived class
class Programmer: public Employee {
  public:
    int bonus;
    void setSalary(int s) {
      salary = s;
    }
    int getSalary() {
      return salary;
    }
};

int main() {
  Programmer myObj;
  myObj.setSalary(50000);
  myObj.bonus = 15000;
  cout << "Salary: " << myObj.getSalary() << "\n";
  cout << "Bonus: " << myObj.bonus << "\n";
  return 0;
}
```

C++ Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance

- Inheritance lets us inherit attributes and methods from another class.
- Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

```
// Base class
class Animal {
  public:
    void animalSound() {
    cout << "The animal makes a sound \n" ;
  }
};

// Derived class
class Pig : public Animal {
  public:
    void animalSound() {
    cout << "The pig says: wee wee \n" ;
```

```
  }
};

// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
    cout << "The dog says: bow wow \n" ;
  }
};

int main() {
  Animal myAnimal;
  Pig myPig;
  Dog myDog;

  myAnimal.animalSound();
  myPig.animalSound();
  myDog.animalSound();
  return 0;
}
```

C++ Files

```
#include <iostream>
#include <fstream>
```

ofstream	Creates and writes to files
ifstream	Reads from files
fstream	A combination of ofstream and ifstream: creates, reads, and writes to files

Create and Write To a File

To create a file, use either the ofstream or fstream class, and specify the name of the file.
To write to the file, use the insertion operator (<<).

```
int main() {
  // Create and open a text file
  ofstream MyFile("filename.txt");

  // Write to the file
  MyFile << "Files can be tricky, but it is fun enough!";

  // Close the file
  MyFile.close();
}
```

Read a File

```
// Create a text string, which is used to output the text file
string myText;

// Read from the text file
ifstream MyReadFile("filename.txt");

// Use a while loop together with the getline() function to read the file line by line
while (getline (MyReadFile, myText)) {
  // Output the text from the file
  cout << myText;
}

// Close the file
MyReadFile.close();
```

C++ Exceptions

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw

an exception (throw an error).

Exception handling in C++ consist of three keywords: try, throw and catch:

- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The throw keyword throws an exception when a problem is detected, which lets us create a custom error.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The try and catch keywords come in pairs:

C++ try and catch

```
try {
  // Block of code to try
  throw exception; // Throw an exception when a problem arise
}
catch () {
  // Block of code to handle errors
}
```

Example:

```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw (age);
  }
}
catch (int myNum) {
  cout << "Access denied - You must be at least 18 years old.\n";
  cout << "Age is: " << myNum;
}
```

Handle Any Type of Exceptions (...)

```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw 505;
  }
}
catch (...) {
  cout << "Access denied - You must be at least 18 years old.\n";
}
```