**C++ Courses**

Introduction

**C++** is a high-level, general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, also referred to as "C with Classes". The language has expanded significantly over time, and modern C++ has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language.

Why Use C++

* C++ is one of the world's most popular programming languages.
* C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.
* C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.
* C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
* C++ is fun and easy to learn
* As C++ is close to C# and Java, it makes it easy for programmers to switch to C++ or vice versa

Hello World

When you first open a C++ IDE, the following program will appear on your screen:

#include <iostream>  
  
*int main() {****std::****cout << "Hello World!";  
  return 0;  
}*

If you build and then run it, the outcome on your console will be:

*Hello World!*

In order to understand this example, it is needed to understand what each line does separately.

**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. 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 some objects:

For example:

*std****::****cout << "Hello World!";*

**Line 4:** Another thing that always appear in a C++ program, is*int main()*. This is called a function. Any code inside its curly brackets {} will be executed.

**Line 5:** *cout* is an object used together with the insertionoperator (<<) to output/print text. In our example it will output "Hello World".

**Line 6:** *return 0;* ends the main function.

Control output and User input

Cout

The *cout* object, together with the << operator, is used to output values/print text, as we previously saw with the *Hello World* example.

Syntax:

*cout<< “Anything you want to print”;*

Cout can be followed by text and/or variables, in whatever order and it can be used as many times as you want in the code.

The cout order does not automatically insert a new line. To do that, you can use the \n character:

For example:

*cout << "Hello World!****\n****";  
cout << "I am learning C++";*

Will have as output:

*Hello World!*

*I am learning C++*

Tip**:** Two \n characters after each other will create a blank line

Another way to insert a new line, is with the endl manipulator.

For example:

*cout << "Hello World!" <<****endl****;  
cout << "I am learning C++";*

Console output:

*Hello World!*

*I am learning C++*

Cin

In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is cin.  
  
For formatted input operations, cin is used together with the extraction operator, which is written as >> (i.e., two "greater than" signs). This operator is then followed by the variable where the extracted data is stored.

For example:

*int number;*

*cin>>number;*

The first statement declares a variable of type int called age, and the second extracts from cin a value to be stored in it. This operation makes the program wait for input from cin; generally, this means that the program will wait for the user to enter some sequence with the keyboard. Once the statement with the extraction operation on cin is reached, the program will wait for as long as needed until some input is introduced.

Tip: Note that the characters introduced using the keyboard are only transmitted to the program when the ENTER (or RETURN) key is pressed.

Extractions on cin can also be chained to request more than one datum in a single statement:

*cin >> a >> b;*

This is equivalent to:

*cin >> a;*

*cin >> b;*In both cases, the user is expected to introduce two values, one for variable a, and another for variable b. Any kind of space is used to separate two consecutive input operations; this may either be a space, a tab, or a new-line character.

The extraction operator can be used on cin to get strings of characters in the same way as with fundamental data types:

*string mystring;*

*cin >> mystring;*  
However, cin extraction always considers spaces (whitespaces, tabs, new-line...) as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an entire sentence.

To get an entire line from cin, there exists a function, called getline, that takes the stream (cin) as first argument, and the string variable as second.

For example:

string mystr;

cout << "What's your name? ";

getline (cin, mystr);

cout << "Hello " << mystr << ".\n";

Will have an output:

What's your name?

(For input: George Papadopoulos)

Hello George Papadopoulos.

Comments

Comments can be used to explain C++ code, and to make it more readable. It can also be used to prevent execution when testing alternative code. Comments can be singled-lined or multi-lined.

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by the compiler (will not be executed).

For example:

*// This is a comment  
cout << "Hello World!";*

Console output:

*Hello World!*

Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by the compiler:

For example:

*/\* The compiler will completely ignore this part,*

*and will go on printing the message Hello World! \*/  
cout << "Hello World!";*

Console output:

*Hello World!*

Variables

In C++, there are different **types** of variables (defined with different keywords), for example:

* *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

To be able to use a variable, you first have to initialize it. To do that, you mention the datatype it belongs to, its name and possible a value.

Syntax

*type* *variable* = *value*;

Where *type* is one of C++ types (such as int), and *variable* is the name of the variable (such as x or myName). The equal sign is used to assign values to the variable.

For example:

*int myAge = 20;  
cout << myAge;*

The example above creates a variable type int by the name of myAge, and stores the value 20.

So the console output will be:

20

Note that if you assign a new value to an existing variable, it will overwrite the previous value.

For example:

int myAge = 19;     
myAge = 20;     
cout << myAge;

Console output:

20

Let’s see how the other datatypes are created:

*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)*

Strings

We mentioned them in the Variables lesson.

Strings are used for storing text.

A string variable contains a collection of characters surrounded by double quotes

To use strings, you must include an additional header file in the source code, the <string> library:

For example:

*// Include the string library  
#include <string>  
  
// Create a string variable  
string greeting = "Hello";*

**String Concatenation**

The + operator can be used between strings to add them together to make a new string.

For example:

*string firstName = "George ";  
string lastName = "Papadopoulos";  
string fullName = firstName + lastName;  
cout << fullName;*

Console output:

*George Papadopoulos*

Tip: Look at the white space after *string firstName = “George “.* Should it not be there, the output would be *GeorgePapadopoulos*.

Although, the space could have been added manually.

For example:

*string firstName = "George";  
string lastName = "Papadopoulos";  
string fullName = firstName + " " + lastName;  
cout << fullName;*

Console output:

*George Papadopoulos*

Or you can also concatenate strings with the *append*() function.

For example:

*string firstName = "George ";  
string lastName = "Papadopoylos";  
string fullName = firstName.append(lastName);  
cout << fullName;*

Console output:

*George Papadopoulos*

**String Length**

To get the length of a string, use the length() function:

Example

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

Console output:

*The length of the txt string is: 26*

You can access the characters in a string by referring to its index number inside square brackets [].

The example below prints the **first character** in **myString**.

Example:

string myString = "Hello";  
cout << myString[0];

Console output:

H

Tip**:** String indexes start with 0: [0] is the first character. [1] is the second character, etc.

**Change String Characters**

To change the value of a specific character in a string, refer to the index number, and use single quotes.

For example:

*string myString = "Hello";  
myString[0] = 'J';  
cout << myString;*

Console output:

*Jello*

Arrays

C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

For your information: All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

In order to use an array, you first have to declare it by specifying the type and number of the elements you are to use.

Syntax:

*type arrayName [ arraySize ];*

This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type.

For example:

*double number[5];*

The expression above declares a 5-element array called balance of type double.

You can initialize C++ array elements either one by one or using a single statement.

For example:

*int number[5] = {10, 20, 30, 40, 50};*

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

For example:

*int randomNumber = number[4];*

The above statement will take 5th element from the array and assign the value to randomNumber variable.

Example:

*int sum = 0;*

*int number[5] = {10, 20, 30, 40, 50};*

*for (int i = 0; i < 5; ++i)*

*{*

*sum += number[i];*

*}*

*cout << "Sum = " << sum << endl;*

Console output:

*Sum = 150*

Conditional Statements

C++ has the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

**The if Statement**

Use the if statement to specify a block of C++ code to be executed if a condition is true.

Syntax:

*if (condition) {  
  // block of code to be executed if the condition is true  
}*

Tip: Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

*if (20 > 18) {  
  cout << "20 is greater than 18";  
}*

Console output:

*20 is greater than 18*

**The else Statement**

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:

*if (condition) {  
  // block of code to be executed if the condition is true  
} else {  
  // block of code to be executed if the condition is false  
}*

For example:

*int age = 20;  
if (age < 18) {  
  cout << "Underage.";  
} else {  
  cout << "Adult.";  
}*

Console output:

*Adult.*

**The else if Statement**

Use the else if statement to specify a new condition if the first condition is false.

Syntax:

*if (condition1) {  
  // block of code to be executed if condition1 is true  
} else if (condition2) {  
  // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
  // block of code to be executed if the condition1 is false and condition2 is false  
}*

For example:

*int age = 22;  
if (age < 13) {  
  cout << "Child.";  
} else if (age < 18) {  
  cout << "Teen.";  
} else {  
  cout << "Adult.";  
}*

Console output:

*Adult.*

**Short Hand If...Else (Ternary Operator)**

There is also a short-hand if else, which is known as the ternary operator because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements.

Syntax:

variable = (condition) ? expressionTrue : expressionFalse;

So instead of writing:

*int age = 20;  
if (age < 18) {  
  cout << "Underage.";  
} else {  
  cout << "Adult.";  
}*

You can simply write:

*int age = 20;  
string result = (age < 18) ? "Underage." : "Adult.";  
cout << result;*

Switch case

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

*switch(expression) {*

*case constant-expression :*

*statement(s);*

*break; //optional*

*case constant-expression :*

*statement(s);*

*break; //optional*

*default : //Optional*

*statement(s);*

*}*

Tip: You can have as many cases as it suits your program.

The following rules apply to a switch statement:

* The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
* The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
* When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
* When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
* Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
* A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

For example:

int grade;

switch(grade) {

*case 'A' :*

*cout << "Excellent!" << endl;*

*break;*

*case 'B' :*

*case 'C' :*

*cout << "Well done" << endl;*

*break;*

*case 'D' :*

*cout << "You passed" << endl;*

*break;*

*case 'F' :*

*cout << "Better try again" << endl;*

*break;*

*default :*

*cout << "Invalid grade" << endl;*

*}*

For grade = ‘A’; , the console output will be:

*Excellent!*

For grade = ‘B’; and grade = ‘C’; , the console output will be:

*Well done!*

For grade = ‘D’; , the console output will be:

*You passed*

For grade = ‘F’; , the console output will be:

*Better try again*

And for any other value the console output will be:

*Invalid grade*

While - loop

A while loop statement repeatedly executes a target statement as long as a given condition is true.

The while loop evaluates the condition.

* If the condition is true, statements inside the body of while loop are evaluated.
* Then, the condition is evaluated again. This process goes on until the condition is false.
* When the condition is false, while loop is terminated.

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value.

Syntax:

*while (condition) {*

*// statement(s)*

*}*

For example:

*int number, i = 1, factorial = 1;*

*while ( i <= number) {*

*factorial \*= i; //factorial = factorial \* i;*

*++i;*

*}*

*cout<<"Factorial of "<< number <<" = "<< factorial;*

If number = 4, the console output will be:

*Factorial of 4 = 24*

For given value of number = 4, the while loop starts executing the code. Here's how while loop works:

1. Initially, i = 1, test expression i <= number is true and factorial becomes 1.
2. Variable i is updated to 2, condition is true, factorial becomes 2.
3. Variable i is updated to 3, condition is true, factorial becomes 6.
4. Variable i is updated to 4, condition is true, factorial becomes 24.
5. Variable i is updated to 5, condition is false and while loop is terminated.

Do – while loop

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time, because the do...while loop checks its condition at the bottom of the loop.

* The statements inside the body of the loop are executed once. Then, the condition is checked.
* If the condition is true, the body of the loop is executed. This process continues until the condition becomes false.
* When the condition is false, do...while loop is terminated.

Syntax:

*do {*

*statement(s);*

*}*

*while( condition );*

For example:

*int number = 5;*

*do {*

*number = number – 1;*

*cout<< number<< endl;*

*}while(number > 0)*

The code above will work as follows:

1. Initially, value of *number* decreases to 4, it is printed with cout, condition is true and the loop continues.
2. Value of *number* decreases to 3, it is printed, condition is true and the loop continues.
3. Value of *number* decreases to 2, it is printed, condition is true and the loop continues.
4. Value of *number* decreases to 1, it is printed, condition is true and the loop continues.
5. Value of *number* decreases to 0, it is printed, condition is false, so the loop breaks.

For loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

*for ( init; condition; increment ) {*

*statement(s);*

*}*

Here is the for loop works:

* The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
* Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
* After the body of the for loop executes, the flow of control jumps back up to the increment statement. This statement can be left blank, as long as a semicolon appears after the condition.
* The condition is evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

For example:

*for( int a = 1; a < = 5; a = a + 1 ) {*

*cout << "value of a: " << a << endl;*

*}*

Console output:

*value of a: 1*

*value of a: 2*

*value of a: 3*

*value of a: 4*

*value of a: 5*

Pointers

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

A pointer variable points to a data type (like int or string) of the same type and is created with the \* operator.

The address of the variable you're working with is assigned to the pointer. As you know every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory.

For example:

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  
  
cout << food << "\n";  
  
cout << &food << "\n";  
  
cout << ptr << "\n";*

Console output:

*Pizza*

*0x6dfed4 (May differ on your computer)*

*0x6dfed4 (May differ on your computer)*

In the example above:

* You create a pointer variable with the name ptr, that **points to** a string variable, by using the asterisk sign \* (string\* ptr).
* Use the & operator to store the memory address of the variable called food, and assign it to the pointer.

Tip: Note that the type of the pointer has to match the type of the variable you're working with.

Reference

A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

References are often confused with pointers but there are three major differences between references and pointers:

* You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
* Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
* A reference must be initialized when it is created. Pointers can be initialized at any time.

You can then think of a reference as a second label attached to that memory location. Therefore, you can access the contents of the variable through either the original variable name or the reference.

For example:

*string lesson = "C++";     
string &course = lesson;*

Now, we can use either the variable name lesson or the reference name course to refer to the lesson variable:

Example

*string lesson = "C++";     
string &course = lesson;*  
*cout << lesson << "\n";    
cout << course << "\n";*

Console output:

*C++*

*C++*

Functions

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main().

A function 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.

To create (often referred to as *declare*) a function, specify the name of the function, followed by parentheses ().

Syntax:

void *myFunction*() {  
  // code to be executed  
}

A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

* Return Type − A function may return a value. The return\_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword void.
* Function Name − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
* Parameters − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
* Function Body − The function body contains a collection of statements that define what the function does.

For example:

*int max(int num1, int num2) {*

*// local variable declaration*

*int result;*

*if (num1 > num2)*

*result = num1;*

*else*

*result = num2;*

*return result;*

*}*

The example above finds the maximum between two given numbers, and returns its via the variable result, which is returned.

**Function Declarations**

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

Syntax:

*return\_type function\_name( parameter list );*

For the previous example with function max(), the function declaration would be:

*int max(int num1, int num2);*

Tip: Parameter names are not important in function declaration only their type is required, so they can be avoided. For example:

*int max(int, int);*

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Classes

A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

For Example:

Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and doors, speed limits, mileage are their properties.

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

Syntax:

*class ClassName*

*{ Access specifier:*

*Data members;*

*Member functions(){}*

*};*

For example:

*class Car*

*{ Private:*

*Int doors;*

*Int model;*

*void start(){};*

*};*

**Declaring Objects**

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

*ClassName ObjectName;*

For example:

*Car bmw;*

**Data members and member functions**

There are two ways to define functions that belong to a class:

* Inside class definition
* Outside class definition

To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class. This is done by specifying the name of the class, followed the scope resolution :: operator, followed by the name of the function.

An example for definition inside the class:

*Class Car*

*{public:*

*void TurnOn(){*

*//code*

*}*

*}*

An example for definition outside of the class:

*Class Car*

*{public:*

*void TurnOff();*

*}*

*Car::TurnOff(){*

*//code*

*}*

The data members and member functions of class can be accessed using the dot(‘.’) operator with the object.

For example:

*bmw.doors = 4;*

*bmw.TurnOn();*

Constructors and Destructors

**Constructor**

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

To create a constructor, you use the same name as the class, followed by parentheses ().

For example:

*class MyClass {        
  public:              
    MyClass() {      //The constructor   
      cout << "Hello World!";  
    }  
};*

Tip: The constructor has the same name as the class, it is always public, and it does not have any return value.

Constructors can also take parameters (just like regular functions), which can be useful for setting initial values for attributes.

Constructors can also be defined outside the class. First, declare the constructor inside the class, and then define it outside of the class by specifying the name of the class, followed by the scope resolution :: operator, followed by the name of the constructor (which is the same as the class).

**Destructor**

A destructor function is called automatically by the compiler when the object goes out of scope because:

* the function ends
* the program ends
* a block containing local variables ends
* a delete operator is called

Destructors differ from the constructors and any other functions because they have same name as the class preceded by a tilde (~) and don’t take any argument and don’t return anything.

For example:

*class MyClass {        
  public:              
    MyClass() ;*

*~MyClass(){ //The distructor*

*//deletion code*

*}*

*};*

Tip: There can only one be destructor in a class.

[www.cplusplus.com/](http://www.cplusplus.com/)

[www.tutorialspoint.com](http://www.tutorialspoint.com)

[www.programiz.com/](http://www.programiz.com/)

[www.w3schools.com/](http://www.w3schools.com/)

[www.geeksforgeeks.org/](https://www.geeksforgeeks.org/)