**C++ Tutorial**

**What is C++?**

C++ is a cross-platform language that can be used to create high-performance applications.

C++ was developed by Bjarne Stroustrup, as an extension to the C language.

C++ gives programmers a high level of control over system resources and memory.

**Why to learn 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!

**About this tutorial – Same as earlier course data**

**Table of Contents:**

**Level 1:**

Installation

Syntax

Variables

Operators

Conditional Statements and Loops

**Level 2:**

Arrays

Pointers

Functions

OOPS Concept – Classes and Objects

Inheritance

**Level 3:**

Namespaces

File Handling

Exception Handling

**Level 1:**

1. **Installation:**

To start using C++, you need two things:

* A text editor, like Notepad, to write C++ code
* A compiler, like GCC, to translate the C++ code into a language that the computer will understand

An IDE (Integrated Development Environment) is used to edit AND compile the code.

Popular IDE's include Code::Blocks, Eclipse, and Visual Studio. These are all free, and they can be used to both edit and debug C++ code.

1. **Syntax**

### Example

#include <iostream>  
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.

**Line 3:** A blank line. C++ ignores white space. But we use it to make the code more readable.

**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 (pronounced "see-out") is an **object** used together with the *insertion operator* (<<) to output/print text. In our example it will output "Hello World".

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

**Line 7:** Do not forget to add the closing curly bracket } to actually end the main function.

The cout object, together with the << operator, is used to output values/print text:

### Example

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

1. **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 create a variable, specify the type and assign it a value:

### Syntax

*type* *variableName* = *value*;

A demonstration of other data types:

### Example

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)

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

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

1. **Operators**

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

### Example

int x = 100 + 50;

C++ divides the operators into the following groups:

* [Arithmetic operators](https://www.w3schools.com/CPP/cpp_operators.asp#arithmetic)
* [Assignment operators](https://www.w3schools.com/CPP/cpp_operators_assignment.asp)
* [Comparison operators](https://www.w3schools.com/CPP/cpp_operators_comparison.asp)
* [Logical operators](https://www.w3schools.com/CPP/cpp_operators_logical.asp)
* Bitwise operators

**Please insert the operator table as per your convenience**

1. **Conditional Statements and Loops**

C++ supports the usual logical conditions from mathematics:

* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b
* Equal to a == b
* Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

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

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*  
*}*

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*  
*}*

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*  
*}*

## C++ Switch Statements

Use the switch statement to select one of many code blocks to be executed.

### Syntax

switch(*expression*) {  
 case x:  
 *// code block*  
break;  
 case y:  
 *// code block*  
break;  
 default:  
 *// code block*  
*}*

This is how it works:

* The switch expression is evaluated once
* The value of the expression is compared with the values of each case
* If there is a match, the associated block of code is executed
* The break and default keywords are optional, and will be described later in this chapter

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.

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

## C++ While Loop

The while loop loops through a block of code as long as a specified condition is true:

### Syntax

while (*condition*) {  
 *// code block to be executed*  
*}*

## The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### Syntax

do {  
 *// code block to be executed*  
*}*  
*while* (*condition*);

## C++ For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

### Syntax

for (*statement 1*; *statement 2*; *statement 3*) {  
 *// code block to be executed*  
*}*

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

## 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.

**Level 2:**

1. **Arrays**

## C++ Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store:

string cars[4];

We have now declared a variable that holds an array of four strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};

## Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the **first element** in **cars**:

### Example

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

## Loop Through an Array

You can loop through the array elements with the for loop.

The following example outputs all elements in the **cars** array:

### Example

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

1. **Pointers**

A **pointer** however, 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:

### 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  
  
// 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";

#### Example explained

Create a pointer variable with the name ptr, that **points to** a string variable, by using the asterisk sign \* (string\* ptr). Note that the type of the pointer has to match the type of the variable you're working with.

Use the & operator to store the memory address of the variable called food, and assign it to the pointer.

Now, ptr holds the value of food's memory address.

1. **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.

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  
}

#### Example Explained

* myFunction() is the name of the function
* void means that the function does not have a return value. You will learn more about return values later in the next chapter
* inside the function (the body), add code that defines what the function should do

## Function Declaration and Definition

A C++ function consist of two parts:

* **Declaration:** the return type, the name of the function, and parameters (if any)
* **Definition:** the body of the function (code to be executed)

void **myFunction()** { // **declaration**  
// the body of the function (**definition**)  
}

## Function Overloading

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

### Example

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

1. **OOPS Concept – Classes and Objects**

## C++ What are Classes and Objects?

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

So, 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++ is an object-oriented programming language.

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: 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

To create a class, use the class keyword:

### Example

Create a class called "MyClass":

class MyClass { // The class  
 public: // Access specifier  
 int myNum; // Attribute (int variable)  
 string myString; // Attribute (string variable)  
};

In C++, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.

To create an object of MyClass, specify the class name, followed by the object name.

To access the class attributes (myNum and myString), use the dot syntax (.) on the object:

### Example

Create an object called "myObj" and access the attributes:

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;  
}

1. **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.

In the example below, the Car class (child) inherits the attributes and methods from the Vehicle class (parent):

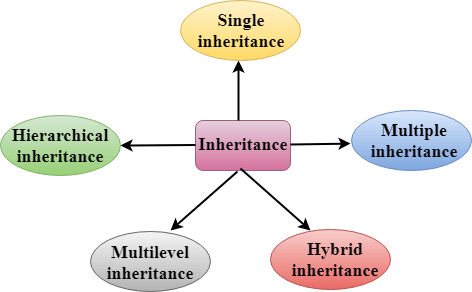
### Example

// 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;  
}

## Types Of Inheritance

**C++ supports five types of inheritance:**21.3M4History of Java

* Single inheritance
* Multiple inheritance
* Hierarchical inheritance
* Multilevel inheritance
* Hybrid inheritance



**Level 3:**

1. **Namespaces**

# C++ Namespaces

Namespaces in C++ are used to organize too many classes so that it can be easy to handle the application.

For accessing the class of a namespace, we need to use namespacename::classname. We can use **using** keyword so that we don't have to use complete name all the time.

In C++, global namespace is the root namespace. The global::std will always refer to the namespace "std" of C++ Framework.

## C++ namespace example: by using keyword

Let's see another example of namespace where we are using "using" keyword so that we don't have to use complete name for accessing a namespace program.

#include <iostream>

**using** **namespace** std;

**namespace** First{

**void** sayHello(){

cout << "Hello First Namespace" << endl;

}

}

**namespace** Second{

**void** sayHello(){

cout << "Hello Second Namespace" << endl;

}

}

**using** **namespace** First;

**int** main () {

sayHello();

**return** 0;

}

Output:

Hello First Namespace

1. **Files and Streams**

C++ Read and Write Example

Let's see the simple example of writing the data to a text file **testout.txt** and then reading the data from the file using C++ FileStream programming.

#include <fstream>

#include <iostream>

**using** **namespace** std;

**int** main () {

**char** input[75];

ofstream os;

os.open("testout.txt");

cout <<"Writing to a text file:" << endl;

cout << "Please Enter your name: ";

cin.getline(input, 100);

os << input << endl;

cout << "Please Enter your age: ";

cin >> input;

cin.ignore();

os << input << endl;

os.close();

ifstream is;

string line;

is.open("testout.txt");

cout << "Reading from a text file:" << endl;

**while** (getline (is,line))

{

cout << line << endl;

}

is.close();

**return** 0;

}

**Output:**

Writing to a text file: Please Enter your name: Nakul Jain Please Enter your age: 22 Reading from a text file: Nakul Jain 22

1. **Exception Handling**

## C++ try and catch

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:

### Example

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

If you do not know the throw **type** used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

### Example

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";  
}