C++

programming language

tutorialspoint

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

This C++ tutorial has been written by experienced C++ programmers. The tutorial helps beginners to advanced programmers in learning C++ in simple and easy steps. This tutorial uses a simple and practical approach to describe the concepts of **C++** to software engineers.

What is C++?

C++ is a middle-level programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs. **C++** runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. C++ is an extension of the C programming language with object-oriented programming concepts. Or, we can say, "C++ is a superset of C programming with additional implementation of object-oriented concepts".

Why to Learn C++?

C++ is an important programming language for students and working professionals to become great software developers. Here are some of the key advantages of learning C++:

- C++ is very close to hardware, so you get a chance to work at a low level, which gives you a lot of control in terms of memory management, better performance, and finally, robust software development.
- **C++ programming** gives you a clear understanding of object-oriented programming. You will understand low level implementation of polymorphism when you implement virtual tables and virtual table pointers, or dynamic type identification.
- C++ is one of the evergreen programming languages and is loved by millions of software developers. If you are a great C++ programmer, then you will never sit without work, and more importantly, you will get highly paid for your work.
- C++ is the most widely used programming language in application and system programming. So you can choose your area of interest in software development.
- C++ really teaches you the difference between compiler, linker, and loader, different data types, storage classes, variable types, their scopes, etc.

There are 1000s of good reasons to learn C++ programming. But one thing is for sure: to learn any programming language, not only C++, you just need to code and code and finally code until you become an expert.

Hello, World! Program Using C++

Just to give you a little excitement about **C++ programming**, I'm going to give you a small conventional C++ Hello World program. You can try it by clicking on "Edit & Run".

Below is the code to print "Hello World" on the screen –

```
#include <iostream>

using namespace std;


// main() is where program execution begins.

int main() {
```

```
    cout << "Hello, World!"; // prints Hello, World!
    return 0;
}
```

C++ Online Compiler

We have provided an easy, user-friendly, and fast C++ online compiler, where you can write, save, run, and share your C++ programs. Click on this link to open it: **C++ Online Compiler**.

Try to click the icon ⚙ to run the following C++ code to print conventional "Hello, World!" using C++ programming.

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

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

There are many C++ compilers available that you can use to compile and run the above-mentioned program:

- Apple C++. Xcode
- Bloodshed Dev-C++
- Clang C++
- Cygwin (GNU C++)
- Mentor Graphics
- MINGW - "Minimalist GNU for Windows"
- GNU CC source
- IBM C++
- Intel C++
- Microsoft Visual C++
- Oracle C++
- HP C++

Features of C++

The following are the features of C++ programming language –

- **C language compatibility**: C++ provides backward compatibility with C; it supports all features of C language.

- **Object-oriented programming**: C++ supports the concepts of OOPs such as objects & classes, encapsulation, data binding, inheritance, and polymorphism.
- **Compiled language:** C++ is a compiler language where the complete code is converted into machine language, which makes it a faster programming language.
- **Standard template library:** C++ provides many data structures and algorithms-related library collections, such as template libraries for contains, iterators, algorithms, etc.
- **Dynamic memory management**: C++ provides two operators **new** and **delete** that help to allocate and deallocate memory blocks dynamically.
- **Exception handling**: C++ provides try, catch, and throw blocks for exception handling, which were not available in C programming.

Applications of C++ Programming

As mentioned before, C++ is one of the most widely used programming languages. It has its presence in almost every area of software development. I'm going to list a few of them here:

- **Application Software Development** - C++ programming has been used in developing almost all the major Operating Systems like Windows, Mac OSX and Linux. Apart from the operating systems, the core part of many browsers, like Mozilla Firefox and Chrome have been written using C++. C++ also has been used in developing the most popular database system called MySQL.
- **Programming Languages Development** - C++ has been used extensively in developing new programming languages like C#, Java, JavaScript, Perl, UNIX's C Shell, PHP, Python, and Verilog, etc.
- **Computation Programming** - C++ is the best friend of scientists because of its fast speed and computational efficiencies.
- **Games Development** - C++ is extremely fast, which allows programmers to do procedural programming for CPU-intensive functions and provides greater control over hardware, because of which it has been widely used in the development of gaming engines.
- **Embedded System** - C++ is being heavily used in developing medical and engineering applications like software for MRI machines, high-end CAD/CAM systems, etc.

This list goes on. There are various areas where software developers are happily using C++ to provide great software. I highly recommend you learn C++ and contribute great software to the community.

Learn C++ By Examples

Practicing C++ examples is the best way to learn C++ programming. All chapters of our C++ tutorial have the related examples with explanation. You can simply go through those examples to understand the concept better.

Jobs or Careers in C++

C++ is a versatile and widely used programming language. Here is the list of some job roles that you can get after learning C++ programming:

- Software Engineer
- Game Developer
- Systems Programmer
- Embedded System Developer
- Robotics Engineer
- Database Developer
- Graphics Programmer

Here is the list of the companies hiring C++ developers:

- Microsoft
- Amazon
- Facebook
- IBM
- Adobe
- Apple
- Google

Target Audience - Who should Learn C++?

This **C++ tutorial** has been prepared for the beginners to help them understand the basics to advanced concepts of the C++ programming language. This tutorial is useful for software and game developers, embedded system developers, system programmers, students, and educators/trainers. After completing this tutorial, you will have a great level of expertise in Python programming, from which you can take yourself to the next level.

Prerequisites to Learn C++

Before you start practicing with various types of examples given in this C++ tutorial, we are making the assumption that you are already aware of the basics of computer programming and computer programming language. You should also be familiar with –

- C++ Compiler
- IDE
- Text Editor

C++ Practice

After completing the C++ tutorial, you can go through these sections to practice the concepts that you have learned:

- C++ Interview Questions
- C++ Online Quiz

- C++ Online Test
- C++ Mock Test

C++ Library Reference

The following list has the complete reference of C++ header files −

- C++ <fstream>
- C++ <iomanip>
- C++ <ios>
- C++ <iosfwd>
- C++ <iostream>
- C++ <istream>
- C++ <ostream>
- C++ <sstream>
- C++ <streambuf>
- C++ <atomic>
- C++ <complex>
- C++ <exception>
- C++ <functional>
- C++ <limits>
- C++ <locale>
- C++ <memory>
- C++ <new>
- C++ <numeric>
- C++ <regex>
- C++ <stdexcept>
- C++ <string>
- C++ <thread>
- C++ <tuple>
- C++ <typeinfo>
- C++ <utility>
- C++ <valarray>

C++ STL Library Reference

The following list has the complete reference of C++ STL libraries −

- C++ <array>
- C++ <bitset>
- C++ <deque>

- C++ <forward_list>
- C++ <list>
- C++ <map>
- C++ <multimap>
- C++ <queue>
- C++ <priority_queue>
- C++ <set>
- C++ <stack>
- C++ <unordered_map>
- C++ <unordered_set>
- C++ <vector>
- C++ <algorithm>
- C++ <iterator>

C++ Revision

For a quick revision of C++ programming, go through these links –

- C++ Quick Guide
- C++ Cheat Sheet
- C++ STL Cheat Sheet

C++ Questions & Answers

Explore the latest C++ questions and answers at C++ questions and answers

FAQs on C++ Tutorial

1. What is the easiest way to learn C++ programming?

You can learn C++ programming by following the chapters of TutorialsPoints's C++ tutorial along with the set of examples. All chapters of this C++ tutorial are detailed and explained with the appropriate examples. You have to be regular while learning C++ and practice the examples on a daily basis.

2. Is prior knowledge of any programming is required to learn C++?

No. You can start learning C++ without having knowledge of any programming language. The C++ tutorial provides the knowledge from scratch.

3. What are the important concepts of C++ programming?

Some of the important concepts of C++ programming are:

- Conditional statements
- Looping
- Arrays

- Structures
- Class and objects
- Inheritance
- Polymorphism
- C++ Standard Template Library

4. How much time is required to learn C++ programming?

It depends on your learning skills; on average, a student can easily learn C++ programming within 2-3 months by doing regular practice.

5. Who developed C++ programming language?

Bjarne Stroustrup developed C++ programming as an extension to the C language. C++ can be considered an advanced version of the C language with object-oriented concepts.

6. What was the original name of C++?

The original name of C++ programming was "C with Classes" because C++ consists of all features of the C language along with the classes.

Copyright & Disclaimer

Table of Contents

C++ Basics

1. C++ Overview: Introduction to C++ Programming Language

C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a **middle-level** language, as it comprises a combination of both high-level and low-level language features.

C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named **C with Classes** but later it was renamed C++ in 1983.

C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Note – A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Object-Oriented Programming

C++ fully supports object-oriented programming, including the five pillars of object-oriented development –

- Classes and Objects
- Encapsulation
- Data hiding
- Inheritance
- Polymorphism

Standard Libraries

Standard C++ consists of three important parts –

- The core language giving all the building blocks including variables, data types, and literals, etc.
- The C++ Standard Library giving a rich set of functions manipulating files, strings, etc.
- The Standard Template Library (STL) giving a rich set of methods manipulating data structures, etc.

The ANSI Standard

The ANSI standard is an attempt to ensure that C++ is portable; that code you write for Microsoft's compiler will compile without errors, using a compiler on a Mac, UNIX, a Windows box, or an Alpha.

The ANSI standard has been stable for a while, and all the major C++ compiler manufacturers support the ANSI standard.

Learning C++

The most important thing while learning C++ is to focus on concepts.

The purpose of learning a programming language is to become a better programmer; that is, to become more effective at designing and implementing new systems and at maintaining old ones.

C++ supports a variety of programming styles. You can write in the style of Fortran, C, Smalltalk, etc., in any language. Each style can achieve its aims effectively while maintaining runtime and space efficiency.

Uses of C++

The uses of C++ are as follows:

- C++ is used by hundreds of thousands of programmers in essentially every application domain.
- C++ is being highly used to write device drivers and other software that rely on direct manipulation of hardware under realtime constraints.
- C++ is widely used for teaching and research because it is clean enough for successful teaching of basic concepts.
- Anyone who has used either an Apple Macintosh or a PC running Windows has indirectly used C++ because the primary user interfaces of these systems are written in C++.

C++ Hello World

Get started learning C++ with the first program by printing "Hello World" on the console –

```
#include <iostream>

using namespace std;


int main() {

   cout << "Hello, World!"; // prints Hello, World!

   return 0;

}
```

Output of the above code is:

```
Hello, World!
```

Advantages of C++

C++ programming language has many advantages over other languages. Some of these advantages are listed as follows –

- **Rich Standard Library**: C++ language provides the users with a rich and useful Standard Template Library (STL). This library has a lot of in-built methods and data structure templates to make coding in this language efficient and quick.

- **OOPS Concepts**: C++ language provides users with Object-Oriented Programming concepts like class, object, abstraction, polymorphism and much more. Hence, it acts as a modified and better version of C programming language.
- **Faster Performance**: C++ language is faster in comparison to other languages like Python, Go, C#, and many more. This makes it very useful in embedded systems and gaming processors.
- **Efficient Compiler**: C++ is a compiled language. C++ compiler is very versatile, and it can accept both procedural programs as well as object oriented programs.
- **Hardware Independent**: C++ language is independent of any hardware or system design. C++ programs work on any system on which a C++/GCC compiler installed and enabled.
- **Large Support Base**: C++ is one of the most widely used programming languages across the globe. It has a vast community of developers and programmers. This can be explored on platforms like Github, Reddit, Discord, DEV, Stack Overflow, and many more.

Disadvantages of C++

C++ programming language also has some disadvantages, which are listed below:

- **Error Detection**: C++ provides the facility of low-level design and is very close to the hardware of the system. Hence, this may lead the user to carry out small errors that are difficult to observe and detect.
- **Large Syntax**: C++ has a very lengthy code base, and many programmers find it difficult to write such a lengthy syntax. This has drawn backlash from the user-base of languages like Python, Go, etc., which are easier to code and simpler to execute.
- **Learning Curve**: As compared to Python and Go, C++ has a very steep learning curve. Users feel that the initial building phase is very tough to learn, and there are many concepts that beginners find difficult to understand.

Facts about C++

Here are some interesting and lesser-known facts about the C++ programming language –

- C++ language was invented at the AT&T Bell Labs, the same place where C language was invented.
- C++ language is heavily used in NASA, where it finds applications in flight software and command design.
- C++ is the successor of the C language. The name C++ has been taken from C only, and the increment operator ('++') signifies that this language is the next version of C.
- C++ is widely used in areas like game development, server-side networking, TCP/IP connections, low-level design, and many more.
- C++ programs begin by executing the **main()** function, and other functions are redirected using the main() function only.
- C++ has inherited almost all features of C, and it has incorporated OOPS concepts from Simula68 programming language.

- C++ does not support pure object-oriented programming. Programs can be executed without the use of classes and objects, just like in procedural languages.
- There are many languages that are conceptualized using C++, and some of those are C#, Java, JavaScript, and many more.

2. C++ Environment Setup

Local Environment Setup

If you are still willing to set up your environment for C++, you need to have the following two software applications on your computer.

Text Editor

This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit Command, Brief, Epsilon, EMACS, and vim or vi.

Name and version of text editor can vary on different operating systems. For example, Notepad will be used on Windows and vim or vi can be used on windows as well as Linux, or UNIX.

The files you create with your editor are called **source files** and for C++, they typically are named with the extension .cpp, .cp, or .c.

A text editor should be in place to start your C++ programming.

C++ Compiler

This is an actual C++ compiler, which will be used to compile your source code into final executable program.

Most C++ compilers don't care what extension you give to your source code, but if you don't specify otherwise, many will use .cpp by default.

Most frequently used and free available compiler is GNU C/C++ compiler, otherwise you can have compilers either from HP or Solaris if you have the respective Operating Systems.

Installing GNU C/C++ Compiler

Here we will discuss the steps of installing GNU C/C++ Compiler on different operating systems.

UNIX/Linux Installation

If you are using **Linux or UNIX** then check whether GCC is installed on your system by entering the following command from the command line –

```
$ g++ -v
```

If you have installed GCC, then it should print a message such as the following –

```
Using built-in specs.

Target: i386-redhat-linux

Configured with: ../configure --prefix=/usr .......

Thread model: posix
```

```
gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)
```

If GCC is not installed, then you will have to install it yourself using the detailed instructions available at https://gcc.gnu.org/install/

Mac OS X Installation

If you use Mac OS X, the easiest way to obtain GCC is to download the Xcode development environment from Apple's website and follow the simple installation instructions.

Xcode is currently available at developer.apple.com/technologies/tools/.

Windows Installation

To install GCC at Windows, you need to install MinGW. To install MinGW, go to the MinGW homepage, www.mingw.org, and follow the link to the MinGW download page. Download the latest version of the MinGW installation program which should be named MinGW-<version>.exe.

While installing MinGW, at a minimum, you must install gcc-core, gcc-g++, binutils, and the MinGW runtime, but you may wish to install more.

Add the bin subdirectory of your MinGW installation to your **PATH** environment variable so that you can specify these tools on the command line by their simple names.

When the installation is complete, you will be able to run gcc, g++, ar, ranlib, dlltool, and several other GNU tools from the Windows command line.

3. C++ Basic Syntax

When we consider a C++ program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what a class, object, method, and instance variable mean:

- **Object** – Objects have states and behaviors. Example: A dog has states (color, name, breed) as well as behaviors (wagging, barking, eating). An object is an instance of a class.
- **Class** – A class can be defined as a template/blueprint that describes the behaviors or states that object of its type supports.
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated, and all the actions are executed.
- **Instance Variables** – Each object has its own unique set of instance variables. An object's state is created by the values assigned to these instance variables.

C++ Program Structure

The basic structure of a C++ program consists of the following parts:

- **Header file inclusion section**: This is the section where we include all required header files whose functions we are going to use in the program.
- **Namespace section**: This is the section where we use the namespace.
- **The main() section**: In this section, we write our main code. The main() function is an entry point of any C++ programming code from where the program's execution starts.

To learn more about it, read: C++ Hello, World Program.

Example

Let us look at a simple code that would print the words *Hello World*.

```
#include <iostream>

using namespace std;


// main() is where program execution begins.

int main() {

   cout << "Hello World"; // prints Hello World

   return 0;

}
```

Example Explanation

Let us look at the various parts of the above program –

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.
- The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.
- The next line '**// main() is where program execution begins.**' is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
- The line **int main()** is the main function where program execution begins.
- The next line **cout << "Hello World";** causes the message "Hello World" to be displayed on the screen.
- The next line **return 0;** terminates main() function and causes it to return the value 0 to the calling process.

Compile and Execute C++ Program

Let's look at how to save the file, compile and run the program. Please follow the steps given below –

- Open a text editor and add the code as above.
- Save the file as: hello.cpp
- Open a command prompt and go to the directory where you saved the file.
- Type 'g++ hello.cpp' and press enter to compile your code. If there are no errors in your code the command prompt will take you to the next line and would generate a.out executable file.
- Now, type 'a.out' to run your program.
- You will be able to see ' Hello World ' printed on the window.

```
$ g++ hello.cpp

$ ./a.out

Hello World
```

Make sure that g++ is in your path and that you are running it in the directory containing file hello.cpp.

You can compile C/C++ programs using makefile. For more details, you can check our 'Makefile Tutorial'.

Semicolons and Blocks in C++

In C++, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

For example, following are three different statements –

```
x = y;

y = y + 1;
```

```
add(x, y);
```

A block is a set of logically connected statements that are surrounded by opening and closing braces. For example –

```
{
   cout << "Hello World"; // prints Hello World
   return 0;
}
```

C++ does not recognize the end of the line as a terminator. For this reason, it does not matter where you put a statement in a line. For example –

```
x = y;
y = y + 1;
add(x, y);
```

is the same as

```
x = y; y = y + 1; add(x, y);
```

C++ Identifiers

A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in C++.

Here are some examples of acceptable identifiers –

```
mohd       zara    abc    move_name  a_123
myname50   _temp   j      a23b9      retVal
```

C++ Keywords

The following list shows the reserved words in C++. These reserved words may not be used as constant or variable or any other identifier names.

asm	Else	new	this
auto	Enum	operator	throw
bool	Explicit	private	true

break	Export	protected	try
case	Extern	public	typedef
catch	False	register	typeid
char	Float	reinterpret_cast	typename
class	For	return	union
const	Friend	short	unsigned
const_cast	Goto	signed	using
continue	If	sizeof	virtual
default	Inline	static	void
delete	Int	static_cast	volatile
do	Long	struct	wchar_t
double	Mutable	switch	while
dynamic_cast	namespace	template	

Trigraphs

A few characters have an alternative representation called a trigraph sequence. A trigraph is a three-character sequence that represents a single character and the sequence always starts with two question marks.

Trigraphs are expanded anywhere they appear, including within string literals and character literals, in comments, and in preprocessor directives.

Following are most frequently used trigraph sequences –

Trigraph	**Replacement**

??=	#
??/	\
??'	^
??(	[
??)	]
??!	|
??<	{
??>	}
??-	~

All the compilers do not support trigraphs and they are not advised to be used because of their confusing nature.

Whitespace in C++

A line containing only whitespace, possibly with a comment, is known as a blank line, and C++ compiler totally ignores it.

Whitespace is the term used in C++ to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins.

Statement 1

```
int age;
```

In the above statement, there must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them.

Statement 2

```
fruit = apples + oranges;   // Get the total fruit
```

In the above statement 2, no whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

C++ Program Structure with Object-oriented Approach

C++ also supports the object-oriented programming approach along with the procedural programming approach.

Example

This example demonstrates the C++ program based on an object-oriented approach.

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

class Numbers {
 private:
  int a;
  int b;

 public:
  // Function to set values
  void setValues(int x, int y) {
    a = x;
    b = y;
  }

  // Function to add these numbers
  double addition() { return a + b; }

  // Function to display values
  void display() { cout << "a: " << a << ", b: " << b << endl; }
};

int main() {
  // Create an object of Numbers class
  Numbers num;

  // Set values
  num.setValues(10, 20);
```

```
  // Display the values
  num.display();

  // Find the addition
  int sum = num.addition();
  cout << "Sum of numbers: " << sum << endl;

  return 0;
}
```

Parts of C++ Program Structure with Object-oriented Approach

The different parts of the C++ program structure with an object-oriented approach are as follows:

1. Class Declaration

A class is a template for an object, or we can say a class is a factory to produce an object. It is a kind of custom data type, where you construct a structure for an object.

A class declaration has the following parts:

- **Access modifiers**: C++ supports three types of access modifiers: ***private***, ***public***, and ***protected***. Accessibilities of the data members and member functions are defined by the access modifiers.
- **Data members and member functions**: The variables used in the class declaration are known as data members, and the member functions are those functions that work on the data members.

Example

As per the above example, the following part of the declaration of a class –

```
class Numbers {
 private:
  int a;
  int b;

 public:
  // Function to set values
  void setValues(int x, int y) {
    a = x;
```

```
    b = y;
  }

  // Function to add these numbers
  double addition() { return a + b; }

  // Function to display values
  void display() { cout << "a: " << a << ", b: " << b << endl; }
};
```

The following are the data members which are defined under the ***private*** access modifier i.e., these data members can be used by the member functions within the class –

```
private:
  int a;
  int b;
```

The following are the member functions used in the class –

```
void setValues(int x, int y);
double addition();
void display();
```

2. Object Creation

In the above example, the following statement is the object creation statement –

```
Numbers num;
```

4. Comments in C++

C++ Comments

Program comments are explanatory statements that you can include in the C++ code. These comments help anyone reading the source code. All programming languages allow different forms of comments.

Types of C++ Comments

C++ supports two types of comments: **single-line comments** and **multi-line comments**. All characters available inside any comment are ignored by the C++ compiler. Here we will discuss these types –

1. C++ Single-line Comments

A single-line comment starts with **//**, extending to the end of the line. These comments can last only till the end of the line, and the next line leads to a new comment.

Syntax

The following syntax shows how to use a single-line comment in C++ –

```
// Text to be commented
```

Example

In the following example, we are creating single-line comments –

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

int main() {
  // this is a single line comment
  cout << "Hello world!" << endl;
  // for a new line, we have to use new comment sections
  cout << "This is second line.";

  return 0;
}
```

Output

The output of the code is –

```
Hello world!
```

```
This is second line.
```

2. C++ Multi-Line Comments

Multi-line comments start with **/*** and end with ***/**. Any text in between these symbols is treated as a comment only.

Syntax

The following syntax shows how to use a multi-line comment in C++ −

```
/* This is a comment */


/*
  C++ comments can also
  span multiple lines
*/
```

Example

In the following example, we are creating multi-line comments −

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

int main() {
  /* Printing hello world!*/
  cout << "Hello World!" << endl;
  /*
  This is a multi-line comment
  Printing another message
  Using cout
  */
  cout << "Tutorials Point";

  return 0;
}
```

Output

The output of the code is −

```
Hello World!
Tutorials Point
```

Comments within Statements

We can also comment-out specific statements within a code block inside a C++ program. This is done using both types of comments.

Example

The following example explains the usage of multi-line comments within statements −

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

int main() {
  cout << "This line" /*what is this*/ << " contains a comment" << endl;
  return 0;
}
```

Output

The output of the code is −

```
This line contains a comment
```

Example

The following example explains the usage of single-line comments within statements −

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

int main() {
  cout << "This line"  // what is this
       << " contains a comment" << endl;
  return 0;
}
```

Output

The output of the code is −

```
This line contains a comment
```

Nesting Comments

Within a **/*** and ***/** comment, **//** characters have no special meaning. Within a **//** comment, **/*** and ***/** have no special meaning. Thus, you can "nest" one kind of comment within the other kind.

Example

The following example explains the usage of comments within comments using nesting −

```
#include <iostream>
using namespace std;
int main() {
  /* Comment out printing of Hello World:

cout << "Hello World"; // prints Hello World

*/
  cout << "New, Hello World!";
  return 0;
}
```

Output

The output of the code is −

```
New, Hello World!
```

Single-line or Multi-Line Comments - When to Use?

Single-line comments are generally used for short lines of comments in general. This is seen in cases where we have to mention a small hint for the algorithm in the code.

Multi-line comments are generally used for longer lines of comments, where the visibility of the whole comment line is necessary. The longer the length of the comment, the more number of statements are needed by the multi-line comments.

Purpose of Comments

Comments are used for various purposes in C++. Some of the main areas of application of comments are as follows −

- To represent a short and concise step in the program for users to understand better.
- To explain a step in a detailed way that is not expressed explicitly in the code.
- To leave different hints for users to grab in the code itself.
- To leave comments for fun or recreation.
- To temporarily disable part of the code for debugging purposes.
- To add metadata to the code for future purposes.
- To create documentation for the code, for example, in Github pages.

5. C++ "Hello, World!" Program

Printing **"Hello, World!"** is the first program in C++. Here, this prints **"Hello, World"** on the console (output screen). To start learning C++, it is the first step to print sometime on the screen.

C++ Program to print "Hello, World!"

Let us see the first C++ program that prints "Hello, World!" –

```
// First C++ program
#include<iostream>
using namespace std;

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

Output

This program will print "Hello, World!" on the output screen. The output will be –

```
Hello, World!
```

Parts of C++ "Hello, World!" Program

Here is the breakdown of the above code and all elements used in the above code –

1. Comment Section (// First C++ program)

Comments are used to specify a textual line that is not supposed to be executed when we compile the code. The compiler ignores the line, and proceeds to the next line. These are used for better readability and explanation of code in the comments section.

This is the comment –

```
// First C++ program
```

2. Preprocessor Directive (#include <iostream>)

The **#include** is known as a pre-processor directive in C++. It is used to include header files with specific methods and elements. Multiple **#include** statements are used to apply different header files in the program. The **iostream** is the header file that defines functions and operations related to the input/output stream.

The statement is used in the program is –

```
#include <iostream>
```

3. Namespace (using namespace std;)

Namespaces are used to differentiate code blocks with the same method names. In this program, the **using namespace std;** is used to set the namespace as standard for users to apply all standard methods in programs.

Here is the code statement used in the program –

```
using namespace std;
```

4. The main() Function (int main(){...})

The **main()** function is the default starting point of any C++ program. It is compulsory for any C++ program to have a main function. The program logics are written inside the main program. The main function body is enclosed inside parenthesis (**{}**).

The **main()** function part is –

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

5. Printing Statement (cout)

The print/output statement is **cout** followed by **"<<"** operator. This is used to print the given parameters specified in the statement on the screen. We can also print multiple elements in a single **cout** block.

The print statement is –

```
cout << "Hello, World!";
```

6. Return Statement (return 0;)

The **return** statement is also known as the exit statement. It is used to exit from the corresponding function. The "**return 0**" is the default statement to exit from the main program.

Here is the return statement used in the program –

```
return 0;
```

Compile and Run "Hello, World!" Program

The "Hello, World!" program can be compiled by using the **Edit & Run** button. You can also open our online C++ compiler, write the program, and compile it there.

The standard way to compile and run the C++ program is explained here: Compile and Run a C++ Program.

==========

End of ebook preview

If you liked what you saw…

Buy it from our store @ https://www.tutorialspoint.com/index.htm