**C++ Overview**

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.

**The C++ Standard Libraries**

The C++ Standard Library is a reference for C++ programmer to help them at every steps of their projects related to system programming. All the C++ functions have been explained in easy to understand way and they can be used easily in your C++ projects.

There is a big list of required header files which can vary depending on different compiler implementations. This header list includes the headers containing the content from the C Standard Library, a list of new C++ specific headers, and other important headers for the C++ Standard Template Library (STL).

The C++ Standard Library can be categorized into two parts −

1. The Standard Function Library − This library consists of general-purpose, stand-alone functions that are not part of any class. The function library is inherited from C.
2. The Object-Oriented Class Library − This is a collection of classes and associated functions.

**The Standard Function Library**

The standard function library is divided into the following categories −

* I/O,
* String and character handling,
* Mathematical,
* Time, date, and localization,
* Dynamic allocation,
* Miscellaneous,
* Wide-character functions,

**The Object-Oriented Class Library**

Standard C++ Object Oriented Library defines an extensive set of classes that provide support for a number of common activities, including I/O, strings, and numeric processing. This library includes the following −

The Standard C++ I/O Classes

* The String Class
* The Numeric Classes
* The STL Container Classes
* The STL Algorithms
* The STL Function Objects
* The STL Iterators
* The STL Allocators
* The Localization library
* Exception Handling Classes
* Miscellaneous Support Library

**Examples are:**

**<algorithm>, <bitset>, <cassert>, <cctype>, <cerrno>, <cfloat>, <ciso646>, <climits>, <clocale>, <cmath>, <complex>, <csetjmp>, <csignal>, <cstdarg>, <cstddef>, <cstdio>, <cstdlib>, <cstring>, <ctime>, <cwchar>, <cwctype>, <deque>, <exception>, <fstream>, <functional>, <hash\_map>, <hash\_set>, <iomanip>, <ios>, <iosfwd>, <iostream>, <istream>, <iterator>, <limits>, <list>, <locale>, <map>, <memory>, <new>, <numeric>, <ostream>, <queue>, <set>, <slist>, <sstream>, <stack>, <stdexcept>, <streambuf>, <string>, <strstream>, <typeinfo>, <utility>, <valarray>, and <vector>.**

**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.

**Use of C++**

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 real-time 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++ 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 on your computer.

**Text Editor**

This will be used to type your program. Examples of few editors include Windows Notepad, Notepad C++, OS Edit command, Brief, Epsilon, EMACS, Editra, 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.

**C++ Basic Syntax**

**C++ Program Structure**

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;*

*}*

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.

**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 |  |

**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.

**Comments in C++**

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 for some form of comments.

C++ supports single-line and multi-line comments. All characters available inside any comment are ignored by C++ compiler.

C++ comments start with /\* and end with \*/. For example −

/\* This is a comment \*/

/\* C++ comments can also

\* span multiple lines

\*/

A comment can also start with //, extending to the end of the line. For example −

*#include <iostream>*

*using namespace std;*

*main() {*

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

*return 0;*

*}*

When the above code is compiled, it will ignore **// prints Hello World** and final executable will produce the following result −

Hello World

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. For example −

/\* Comment out printing of Hello World:

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

\*/