



# LEARN C++

programming language

**tutorialspoint**  
SIMPLY EASY LEARNING

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



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

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.

This tutorial adopts a simple and practical approach to describe the concepts of C++.

## Audience

This tutorial has been prepared for the beginners to help them understand the basic to advanced concepts related to C++.

## Prerequisites

Before you start practicing with various types of examples given in this tutorial, we are making an assumption that you are already aware of the basics of computer program and computer programming language.

## Copyright & Disclaimer

© Copyright 2014 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

<b>About the Tutorial.....</b>	i
<b>Audience .....</b>	i
<b>Prerequisites .....</b>	i

<b>Copyright &amp; Disclaimer.....</b>	i
<b>Table of Contents .....</b>	i
<b>1. OVERVIEW.....</b>	1
<b>Object-Oriented Programming .....</b>	1
<b>Standard Libraries .....</b>	1
<b>The ANSI Standard .....</b>	1
<b>Learning C++.....</b>	2
<b>Use of C++ .....</b>	2
<b>2. ENVIRONMENT SETUP.....</b>	3
<b>Try it Option Online.....</b>	3
<b>Local Environment Setup.....</b>	3
<b>Installing GNU C/C++ Compiler:.....</b>	4
<b>3. BASIC SYNTAX.....</b>	6
<b>C++ Program Structure:.....</b>	6
<b>Compile &amp; Execute C++ Program:.....</b>	7
<b>Semicolons &amp; Blocks in C++ .....</b>	7
<b>C++ Identifiers .....</b>	8
<b>C++ Keywords.....</b>	8
<b>Trigraphs .....</b>	9
<b>Whitespace in C++.....</b>	10
<b>4. COMMENTS IN C++ .....</b>	11
<b>5. DATA TYPES.....</b>	13
<b>Primitive Built-in Types .....</b>	13
<b>typedef Declarations .....</b>	15

<b>Enumerated Types .....</b>	<b>16</b>
<b>6. VARIABLE TYPES .....</b>	<b>17</b>
<b>Variable Definition in C++.....</b>	<b>17</b>
<b>Variable Declaration in C++ .....</b>	<b>18</b>
<b>Lvalues and Rvalues .....</b>	<b>20</b>
<b>7. VARIABLE SCOPE .....</b>	<b>21</b>
<b>Local Variables .....</b>	<b>21</b>
<b>Global Variables .....</b>	<b>22</b>
<b>Initializing Local and Global Variables .....</b>	<b>23</b>
<b>8. CONSTANTS/LITERALS .....</b>	<b>24</b>
<b>Integer Literals .....</b>	<b>24</b>
<b>Floating-point Literals .....</b>	<b>24</b>
<b>Boolean Literals.....</b>	<b>25</b>
<b>Character Literals .....</b>	<b>25</b>
<b>String Literals .....</b>	<b>26</b>
<b>Defining Constants .....</b>	<b>27</b>
<b>9. MODIFIER TYPES.....</b>	<b>29</b>
<b>Type Qualifiers in C++.....</b>	<b>30</b>
<b>10. STORAGE CLASSES.....</b>	<b>31</b>
<b>The auto Storage Class .....</b>	<b>31</b>
<b>The register Storage Class .....</b>	<b>31</b>
<b>The static Storage Class .....</b>	<b>31</b>
<b>The extern Storage Class .....</b>	<b>33</b>
<b>The mutable Storage Class .....</b>	<b>34</b>
<b>11. OPERATORS.....</b>	<b>35</b>

Arithmetic Operators .....	35
Relational Operators .....	37
Logical Operators .....	40
Bitwise Operators .....	41
Assignment Operators.....	44
Misc Operators.....	47
Operators Precedence in C++ .....	48
<b>12. LOOP TYPES.....</b>	<b>51</b>
While Loop.....	52
for Loop.....	54
do...while Loop.....	56
nested Loops.....	58
Loop Control Statements.....	60
Break Statement .....	61
continue Statement .....	63
goto Statement .....	65
The Infinite Loop .....	67
<b>13. DECISION-MAKING STATEMENTS .....</b>	<b>69</b>
If Statement .....	70
if...else Statement .....	72
if...else if...else Statement.....	73
Switch Statement.....	75
Nested if Statement .....	78
The ? : Operator .....	81
<b>14. FUNCTIONS .....</b>	<b>82</b>
Defining a Function .....	82

Function Declarations .....	83
Calling a Function .....	84
Function Arguments.....	85
Call by Value .....	86
Call by Pointer.....	87
Call by Reference.....	89
Default Values for Parameters .....	90
15. NUMBERS.....	93
Defining Numbers in C++.....	93
Math Operations in C++ .....	94
Random Numbers in C++.....	96
16. ARRAYS.....	98
Declaring Arrays.....	98
Initializing Arrays .....	98
Accessing Array Elements.....	99
Arrays in C++ .....	100
Pointer to an Array.....	103
Passing Arrays to Functions.....	105
Return Array from Functions.....	107
17. STRINGS.....	111
The C-Style Character String .....	111
The String Class in C++.....	114
18. POINTERS .....	116
What are Pointers? .....	116
Using Pointers in C++.....	117
Pointers in C++ .....	118

Null Pointers .....	119
Pointer Arithmetic.....	120
Pointers vs Arrays .....	124
Array of Pointers .....	126
Pointer to a Pointer.....	128
Passing Pointers to Functions.....	130
Return Pointer from Functions .....	132
<b>19. REFERENCES .....</b>	<b>135</b>
References vs Pointers .....	135
Creating References in C++.....	135
References as Parameters .....	137
Reference as Return Value .....	138
<b>20. DATE AND TIME.....</b>	<b>141</b>
Current Date and Time .....	142
Format Time using struct tm .....	143
<b>21. BASIC INPUT/OUTPUT .....</b>	<b>145</b>
I/O Library Header Files.....	145
The Standard Output Stream (cout) .....	145
The Standard Input Stream (cin).....	146
The Standard Error Stream (cerr) .....	147
The Standard Log Stream (clog).....	148
<b>22. DATA STRUCTURES.....</b>	<b>149</b>
Defining a Structure .....	149
Accessing Structure Members .....	150

Structures as Function Arguments.....	151
Pointers to Structures .....	153
The <b>typedef</b> Keyword .....	155
<b>23. CLASSES AND OBJECTS .....</b>	<b>157</b>
<b>C++ Class Definitions .....</b>	<b>157</b>
Define C++ Objects .....	157
Accessing the Data Members .....	158
Classes & Objects in Detail .....	159
Class Access Modifiers.....	163
The <b>public</b> Members.....	164
The <b>private</b> Members.....	165
The <b>protected</b> Members.....	167
Constructor & Destructor .....	169
Parameterized Constructor .....	170
The Class Destructor.....	173
Copy Constructor .....	174
Friend Functions.....	179
Inline Functions.....	181
this Pointer .....	182
Pointer to C++ Classes .....	184
Static Members of a Class .....	185
Static Function Members .....	187
<b>24. INHERITANCE.....</b>	<b>190</b>
Base & Derived Classes.....	190
Access Control and Inheritance .....	192

Type of Inheritance .....	192
Multiple Inheritance .....	193
<b>25. OVERLOADING (OPERATOR &amp; FUNCTION).....</b>	<b>196</b>
Function Overloading in C++ .....	196
Operators Overloading in C++ .....	197
Overloadable/Non-overloadable Operators.....	200
Operator Overloading Examples .....	201
Unary Operators Overloading .....	201
Increment (++) and Decrement (- -) Operators .....	203
Binary Operators Overloading.....	205
Relational Operators Overloading .....	208
Input/Output Operators Overloading.....	210
++ and -- Operators Overloading .....	212
Assignment Operators Overloading .....	214
Function Call () Operator Overloading .....	215
Subscripting [ ] Operator Overloading .....	217
Class Member Access Operator - > Overloading .....	219
<b>26. POLYMORPHISM.....</b>	<b>223</b>
Virtual Function .....	226
Pure Virtual Functions.....	226
<b>27. DATA ABSTRACTION .....</b>	<b>227</b>
Access Labels Enforce Abstraction.....	228
Benefits of Data Abstraction .....	228
Data Abstraction Example .....	228
Designing Strategy .....	230
<b>28. DATA ENCAPSULATION.....</b>	<b>231</b>

Data Encapsulation Example .....	232
Designing Strategy .....	233
<b>29. INTERFACES.....</b>	<b>234</b>
Abstract Class Example .....	234
Designing Strategy .....	236
<b>30. FILES AND STREAMS.....</b>	<b>238</b>
Opening a File .....	238
Closing a File .....	239
Writing to a File.....	239
Reading from a File .....	239
Read & Write Example .....	240
File Position Pointers.....	242
<b>31. EXCEPTION HANDLING.....</b>	<b>243</b>
Throwing Exceptions .....	244
Catching Exceptions .....	244
C++ Standard Exceptions .....	246
Define New Exceptions .....	247
<b>32. DYNAMIC MEMORY.....</b>	<b>249</b>
The new and delete Operators.....	249
Dynamic Memory Allocation for Arrays .....	251
Dynamic Memory Allocation for Objects.....	251
<b>33. NAMESPACES .....</b>	<b>253</b>
Defining a Namespace.....	253
The using directive .....	254

Discontiguous Namespaces .....	256
Nested Namespaces .....	256
34. TEMPLATES .....	258
Function Template .....	258
Class Template .....	259
35. PREPROCESSOR .....	263
The #define Preprocessor.....	263
Function-Like Macros .....	264
Conditional Compilation .....	264
The # and ## Operators .....	266
Predefined C++ Macros .....	268
36. SIGNAL HANDLING .....	270
The signal() Function .....	270
The raise() Function.....	272
37. MULTITHREADING.....	274
Creating Threads .....	274
Terminating Threads .....	275
Passing Arguments to Threads .....	277
Joining and Detaching Threads.....	278
38. WEB PROGRAMMING.....	282
What is CGI?.....	282
Web Browsing .....	282
CGI Architecture Diagram.....	282
Web Server Configuration .....	283
First CGI Program .....	284

<b>My First CGI program .....</b>	<b>284</b>
<b>HTTP Header .....</b>	<b>285</b>
<b>CGI Environment Variables.....</b>	<b>285</b>
<b>C++ CGI Library.....</b>	<b>289</b>
<b>GET and POST Methods.....</b>	<b>289</b>
<b>Passing Information Using GET Method .....</b>	<b>289</b>
<b>Simple URL Example: Get Method.....</b>	<b>290</b>
<b>Simple FORM Example: GET Method.....</b>	<b>291</b>
<b>Passing Information Using POST Method .....</b>	<b>292</b>
<b>Passing Checkbox Data to CGI Program .....</b>	<b>292</b>
<b>Passing Radio Button Data to CGI Program .....</b>	<b>294</b>
<b>Passing Text Area Data to CGI Program .....</b>	<b>296</b>
<b>Passing Dropdown Box Data to CGI Program.....</b>	<b>298</b>
<b>Using Cookies in CGI .....</b>	<b>299</b>
<b>How It Works .....</b>	<b>299</b>
<b>Setting up Cookies.....</b>	<b>300</b>
<b>Retrieving Cookies.....</b>	<b>301</b>
<b>File Upload Example .....</b>	<b>303</b>
<b>39. STL TUTORIAL.....</b>	<b>306</b>
<b>40. STANDARD LIBRARY.....</b>	<b>309</b>
<b>The Standard Function Library .....</b>	<b>309</b>
<b>The Object Oriented Class Library .....</b>	<b>309</b>

# 1. 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.

**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 four pillars of object-oriented development:

- 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.

#### 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++.

## 2. ENVIRONMENT SETUP

### Try it Option Online

You really do not need to set up your own environment to start learning C++ programming language. Reason is very simple, we have already set up C++ Programming environment online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try the following example using our online compiler option available at <http://www.compileonline.com/>

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

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

For most of the examples given in this tutorial, you will find **Try it** option in our website code sections at the top right corner that will take you to the online compiler. So just make use of it and enjoy your learning.

### Local Environment Setup

If you are still willing to set up your environment for C++, you need to have the following two softwares 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:

## 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 <http://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/](http://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](http://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. 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, methods, and instant variables mean.

- **Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, and eating. An object is an instance of a class.
- **Class** - A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- **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.
- **Instant Variables** - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

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:

1. 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.
2. The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.

3. 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.
4. The line **int main()** is the main function where program execution begins.
5. The next line **cout << "This is my first C++ program."**; causes the message "This is my first C++ program" to be displayed on the screen.
6. The next line **return 0;** terminates main() function and causes it to return the value 0 to the calling process.

#### Compile & Execute C++ Program:

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

1. Open a text editor and add the code as above.
2. Save the file as: hello.cpp
3. Open a command prompt and go to the directory where you saved the file.
4. 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.
5. Now, type 'a.out' to run your program.
6. 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 & 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.

# 4. 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  
*/
```

# 5. DATA TYPES

While writing program in any language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

## Primitive Built-in Types

C++ offers the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types:

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Several of the basic types can be modified using one or more of these type modifiers:

- signed
- unsigned
- short
- long

The following table shows the variable type, how much memory it takes to store the value in memory, and what is maximum and minimum value which can be stored in such type of variables.

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,647 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

The size of variables might be different from those shown in the above table, depending on the compiler and the computer you are using.

Following is the example, which will produce correct size of various data types on your computer.

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

int main()
{
    cout << "Size of char : " << sizeof(char) << endl;
    cout << "Size of int : " << sizeof(int) << endl;
    cout << "Size of short int : " << sizeof(short int) << endl;
    cout << "Size of long int : " << sizeof(long int) << endl;
    cout << "Size of float : " << sizeof(float) << endl;
    cout << "Size of double : " << sizeof(double) << endl;
    cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;
    return 0;
}
```

This example uses **endl**, which inserts a new-line character after every line and **<<** operator is being used to pass multiple values out to the screen. We are also using **sizeof()** function to get size of various data types.

When the above code is compiled and executed, it produces the following result which can vary from machine to machine:

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

#### typedef Declarations

You can create a new name for an existing type using **typedef**. Following is the simple syntax to define a new type using **typedef**:

```
typedef type newname;
```

For example, the following tells the compiler that **feet** is another name for **int**:

```
typedef int feet;
```

Now, the following declaration is perfectly legal and creates an integer variable called distance:

```
feet distance;
```

### Enumerated Types

An enumerated type declares an optional type name and a set of zero or more identifiers that can be used as values of the type. Each enumerator is a constant whose type is the enumeration.

Creating an enumeration requires the use of the keyword **enum**. The general form of an enumeration type is:

```
enum enum-name { list of names } var-list;
```

Here, the enum-name is the enumeration's type name. The list of names is comma separated.

For example, the following code defines an enumeration of colors called colors and the variable c of type color. Finally, c is assigned the value "blue".

```
enum color { red, green, blue } c;
c = blue;
```

By default, the value of the first name is 0, the second name has the value 1, and the third has the value 2, and so on. But you can give a name a specific value by adding an initializer. For example, in the following enumeration, **green** will have the value 5.

```
enum color { red, green=5, blue };
```

Here, **blue** will have a value of 6 because each name will be one greater than the one that precedes it.

# 6. VARIABLE TYPES

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive:

There are following basic types of variable in C++ as explained in last chapter:

Type	Description
bool	Stores either value true or false.
char	Typically a single octet (one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.
wchar_t	A wide character type.

C++ also allows to define various other types of variables, which we will cover in subsequent chapters like **Enumeration**, **Pointer**, **Array**, **Reference**, **Data structures**, and **Classes**.

Following section will cover how to define, declare and use various types of variables.

## Variable Definition in C++

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid C++ data type including char, w\_char, int, float, double, bool or any user-defined object, etc., and **variable\_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Some examples are:

```
extern int d = 3, f = 5;      // declaration of d and f.
int d = 3, f = 5;            // definition and initializing d and f.
byte z = 22;                 // definition and initializes z.
char x = 'x';                // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.

#### Variable Declaration in C++

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use **extern** keyword to declare a variable at any place. Though you can declare a variable multiple times in your C++ program, but it can be defined only once in a file, a function or a block of code.

#### Example: