**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. This **C++** tutorial adopts a simple and practical approach to describe the concepts of **C++** for beginners to advanded software engineers.

## Why to Learn C++

**C++** is a MUST for students and working professionals to become a great Software Engineer. I will list down 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 lot of control in terms of memory management, better performance and finally a robust software development.
* **C++ programming** gives you a clear understanding about Object Oriented Programming. You will understand low level implementation of polymorphism when you will implement virtual tables and virtual table pointers, or dynamic type identification.
* C++ is one of the every green programming languages and 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 languages in application and system programming. So you can choose your area of interest of 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 for sure, to learn any programming language, not only C++, you just need to code, and code and finally code until you become expert.

## Hello World 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 using Demo link

C++ is a super set of C programming with additional implementation of object-oriented concepts.

[Live Demo](http://tpcg.io/n4BVuS)

#include <iostream>

using namespace std;

// main() is where program execution begins.

int main() {

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

return 0;

}

There are many C++ compilers available which you can use to compile and run 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++

It is really impossible to give a complete list of all the available compilers. The C++ world is just too large and too much new is happening.

## Applications of C++ Programming

As mentioned before, C++ is one of the most widely used programming languages. It has it's presence in almost every area of software development. I'm going to list 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 and Python, and Verilog etc.
* **Computation Programming** - C++ is the best friends of scientists because of 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 development of gaming engines.
* **Embedded System**- C++ is being heavily used in developing Medical and Engineering Applications like softwares 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 softwares. I highly recommend you to learn C++ and contribute great softwares to the community.

## Audience

This C++ 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 C++ tutorial,we are making an assumption that you are already aware of the basics of computer program and computer programming language.

# 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.

**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 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++ Environment Setup

## 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 <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/](https://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.

# 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, 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, 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.
* **Instance Variables** − Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

## C++ Program Structure

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

[Live Demo](http://tpcg.io/n4BVuS)

#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.

## 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'](https://www.tutorialspoint.com/makefile/index.htm).

## 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.

# 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 −

[Live Demo](http://tpcg.io/Q4esaC)

#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

\*/

# C++ 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 | 2bytes | 0 to 65,535 |
| signed short int | 2bytes | -32768 to 32767 |
| long int | 8bytes | -2,147,483,648 to 2,147,483,647 |
| signed long int | 8bytes | same as long int |
| unsigned long int | 8bytes | 0 to 4,294,967,295 |
| long long int | 8bytes | -(2^63) to (2^63)-1 |
| unsigned long long int | 8bytes | 0 to 18,446,744,073,709,551,615 |
| float | 4bytes |  |
| double | 8bytes |  |
| long double | 12bytes |  |
| 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.

[Live Demo](http://tpcg.io/iKNn78)

#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()** operator 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.

# C++ 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 −

|  |  |
| --- | --- |
| **Sr.No** | **Type & Description** |
| 1 | **bool**  Stores either value true or false. |
| 2 | **char**  Typically a single octet (one byte). This is an integer type. |
| 3 | **int**  The most natural size of integer for the machine. |
| 4 | **float**  A single-precision floating point value. |
| 5 | **double**  A double-precision floating point value. |
| 6 | **void**  Represents the absence of type. |
| 7 | **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 definition 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

Try the following example where a variable has been declared at the top, but it has been defined inside the main function −

[Live Demo](http://tpcg.io/odXXRO)

#include <iostream>

using namespace std;

// Variable declaration:

extern int a, b;

extern int c;

extern float f;

int main () {

// Variable definition:

int a, b;

int c;

float f;

// actual initialization

a = 10;

b = 20;

c = a + b;

cout << c << endl ;

f = 70.0/3.0;

cout << f << endl ;

return 0;

}

When the above code is compiled and executed, it produces the following result −

30

23.3333

Same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For example −

// function declaration

int func();

int main() {

// function call

int i = func();

}

// function definition

int func() {

return 0;

}

## Lvalues and Rvalues

There are two kinds of expressions in C++ −

* **lvalue** − Expressions that refer to a memory location is called "lvalue" expression. An lvalue may appear as either the left-hand or right-hand side of an assignment.
* **rvalue** − The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement −

int g = 20;

But the following is not a valid statement and would generate compile-time error −

10 = 20;

# Variable Scope in C++

A scope is a region of the program and broadly speaking there are three places, where variables can be declared −

* Inside a function or a block which is called local variables,
* In the definition of function parameters which is called formal parameters.
* Outside of all functions which is called global variables.

We will learn what is a function and it's parameter in subsequent chapters. Here let us explain what are local and global variables.

## Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables −

[Live Demo](http://tpcg.io/QIjnPh)

#include <iostream>

using namespace std;

int main () {

// Local variable declaration:

int a, b;

int c;

// actual initialization

a = 10;

b = 20;

c = a + b;

cout << c;

return 0;

}

## Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables −

[Live Demo](http://tpcg.io/dRHHpD)

#include <iostream>

using namespace std;

// Global variable declaration:

int g;

int main () {

// Local variable declaration:

int a, b;

// actual initialization

a = 10;

b = 20;

g = a + b;

cout << g;

return 0;

}

A program can have same name for local and global variables but value of local variable inside a function will take preference. For example −

[Live Demo](http://tpcg.io/dt7MP9)

#include <iostream>

using namespace std;

// Global variable declaration:

int g = 20;

int main () {

// Local variable declaration:

int g = 10;

cout << g;

return 0;

}

When the above code is compiled and executed, it produces the following result −

10

## Initializing Local and Global Variables

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows −

|  |  |
| --- | --- |
| **Data Type** | **Initializer** |
| int | 0 |
| char | '\0' |
| float | 0 |
| double | 0 |
| pointer | NULL |

It is a good programming practice to initialize variables properly, otherwise sometimes program would produce unexpected result.

# C++ Constants/Literals

Constants refer to fixed values that the program may not alter and they are called **literals**.

Constants can be of any of the basic data types and can be divided into Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

Again, constants are treated just like regular variables except that their values cannot be modified after their definition.

## Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals −

212 // Legal

215u // Legal

0xFeeL // Legal

078 // Illegal: 8 is not an octal digit

032UU // Illegal: cannot repeat a suffix

Following are other examples of various types of Integer literals −

85 // decimal

0213 // octal

0x4b // hexadecimal

30 // int

30u // unsigned int

30l // long

30ul // unsigned long

## Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals −

3.14159 // Legal

314159E-5L // Legal

510E // Illegal: incomplete exponent

210f // Illegal: no decimal or exponent

.e55 // Illegal: missing integer or fraction

## Boolean Literals

There are two Boolean literals and they are part of standard C++ keywords −

* A value of **true** representing true.
* A value of **false** representing false.

You should not consider the value of true equal to 1 and value of false equal to 0.

## Character Literals

Character literals are enclosed in single quotes. If the literal begins with L (uppercase only), it is a wide character literal (e.g., L'x') and should be stored in **wchar\_t** type of variable . Otherwise, it is a narrow character literal (e.g., 'x') and can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C++ when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t). Here, you have a list of some of such escape sequence codes −

|  |  |
| --- | --- |
| **Escape sequence** | **Meaning** |
| \\ | \ character |
| \' | ' character |
| \" | " character |
| \? | ? character |
| \a | Alert or bell |
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \ooo | Octal number of one to three digits |
| \xhh . . . | Hexadecimal number of one or more digits |

Following is the example to show a few escape sequence characters −

[Live Demo](http://tpcg.io/8RqQ5S)

#include <iostream>

using namespace std;

int main() {

cout << "Hello\tWorld\n\n";

return 0;

}

When the above code is compiled and executed, it produces the following result −

Hello World

## String Literals

String literals are enclosed in double quotes. A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separate them using whitespaces.

Here are some examples of string literals. All the three forms are identical strings.

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

## Defining Constants

There are two simple ways in C++ to define constants −

* Using **#define** preprocessor.
* Using **const** keyword.

## The #define Preprocessor

Following is the form to use #define preprocessor to define a constant −

#define identifier value

Following example explains it in detail −

[Live Demo](http://tpcg.io/N6xxDP)

#include <iostream>

using namespace std;

#define LENGTH 10

#define WIDTH 5

#define NEWLINE '\n'

int main() {

int area;

area = LENGTH \* WIDTH;

cout << area;

cout << NEWLINE;

return 0;

}

When the above code is compiled and executed, it produces the following result −

50

## The const Keyword

You can use **const** prefix to declare constants with a specific type as follows −

const type variable = value;

Following example explains it in detail −

[Live Demo](http://tpcg.io/IA1cea)

#include <iostream>

using namespace std;

int main() {

const int LENGTH = 10;

const int WIDTH = 5;

const char NEWLINE = '\n';

int area;

area = LENGTH \* WIDTH;

cout << area;

cout << NEWLINE;

return 0;

}

When the above code is compiled and executed, it produces the following result −

50

Note that it is a good programming practice to define constants in CAPITALS.