C++ Tutorial

C++ tutorial provides basic and advanced concepts of C++. Our C++ tutorial is designed for beginners and professionals.

C++ is an object-oriented programming language. It is an extension to C programming.

Our C++ tutorial includes all topics of C++ such as first example, control statements, objects and classes, inheritance, constructor, destructor, this, static, polymorphism, abstraction, abstract class, interface, namespace, encapsulation, arrays, strings, exception handling, File IO, etc.

What is C++

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

C++ is a middle-level language, as it encapsulates both high and low level language features.

Object-Oriented Programming (OOPs)

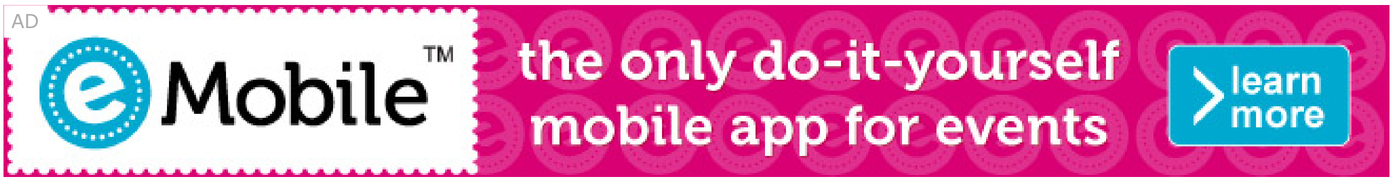
C++ supports the object-oriented programming, the four major pillar of object-oriented programming (OOPs) used in C++ are:

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

C++ Standard Libraries

Standard C++ programming is divided into three important parts:

- The core library includes the data types, variables and literals, etc.
- The standard library includes the set of functions manipulating strings, files, etc.
- The Standard Template Library (STL) includes the set of methods manipulating a data structure.

AD


Usage of C++

By the help of C++ programming language, we can develop different types of secured and robust applications:

- Window application
- Client-Server application
- Device drivers
- Embedded firmware etc

AD

C++ Program

In this tutorial, all C++ programs are given with C++ compiler so that you can easily change the C++ program code.

File: main.cpp

```
#include <iostream>
using namespace std;
int main() {
   cout << "Hello C++ Programming";
   return 0;
}
```

A detailed explanation of first C++ program is given in next chapters.

C++ Index

- algorithm move() function
- algorithm all_of() function
- algorithm copy_backward() function
- algorithm copy_n() function
- algorithm search() function
- algorithm is_permutation() function
- algorithm mismatch() function
- algorithm move_backward() function
- algorithm none_of() function
- algorithm search_n() function
- algorithm swap() function
- algorithm fill() function
- algorithm iter_swap() function
- algorithm replace_copy_if() function
- algorithm replace_copy() function
- algorithm replace_if() function
- algorithm replace() function

- math scalbln() function
- math ilogb() function
- math copysign() function
- math nextafter() function
- math nexttoward() function
- math fdim() function
- math fmax() function
- math fmin() function
- math pow() function
- math sqrt() function
- math cbrt() function
- math hypot() function
- math ceil() function
- math floor() function
- math round() function
- math lround() function
- math llround() function
- math fmod() function
- math trunc() function
- math rint() function
- math lrint() function
- math llrint() function
- math nearbyint() function
- math remainder() function
- math remquo() function
- math fabs() function
- math abs() function

- set equal_range() function
- set get_allocator() function
- set operator==
- set operator!=
- set operator<
- set operator<=
- set operator>
- set operator>=
- set swap() function

C++ STD Strings

- string compare() function
- string length() function
- string swap() function
- string substr() function
- string size() function
- string resize() function
- string replace() function
- string append() function
- string at() function
- string find() function
- string find_first_of() function
- string find_first_not_of() function
- string find_last_of() function
- string find_last_not_of() function
- string insert() function

- algorithm swap_ranges() function
- algorithm transform() function
- algorithm fill_n() function
- algorithm generate_n() function
- algorithm generate() function
- algorithm remove() function
- algorithm is_partitioned() function
- algorithm random_shuffle() function
- algorithm remove_copy_if() function
- algorithm remove_copy() function
- algorithm partition_copy() function
- algorithm partition_point() function
- algorithm partition() function
- algorithm remove_if() function
- algorithm reverse_copy() function
- algorithm reverse() function

- math fma() function
- math fpclassify() function
- math isfinite() function
- math isinf() function
- math isnan() function
- math isnormal() function
- math signbit() function
- math isgreater() function
- math isgreaterequal() function
- math less() function
- math islessequal() function
- math islessgreater() function
- math isunordered() function
- math erf() function
- math erfc() function
- math tgamma() function
- math lgamma() function

C++ STL priority_queue

- C++ priority_queue
- priority_queue emplace() function
- priority_queue empty() function
- priority_queue pop() function

- string max_size() function
- string push_back() function
- string pop_back() function
- sstring assign() function
- string copy() function
- string back() function
- string begin() function
- string capacity() function
- string cbegin() function
- string cend() function
- string clear() function
- string crbegin() function
- string data() function
- string empty() function
- string erase() function
- string front() function
- string operator+=()
- string operator=()
- string operator[]()
- string rfind() function
- string end() function
- string rend() function
- string shrink_to_fit() function
- string c_str() function
- string crend() function
- string rbegin() function
- string reserve() function

Prerequisite

Before learning C++, you must have the basic knowledge of C.

Audience

Our C++ tutorial is designed to help beginners and professionals.

Problem

We assure that you will not find any problem in this C++ tutorial. But if there is any mistake, please post the problem in contact form.

Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk tutorial SPSS tutorial

Difference between C and C++

What is C?

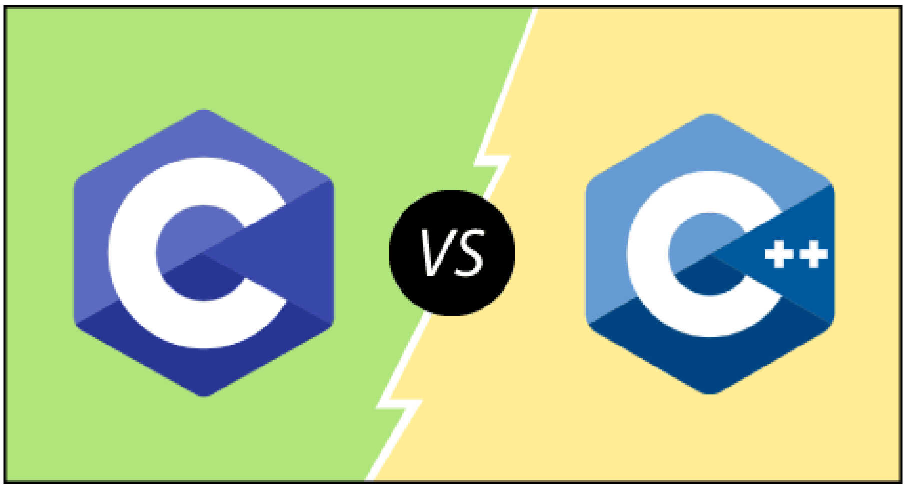
C is a structural or procedural oriented programming language which is machine-independent and extensively used in various applications.

C is the basic programming language that can be used to develop from the operating systems (like Windows) to complex programs like Oracle database, Git, Python interpreter, and many more. C programming language can be called a god's programming language as it forms the base for other programming languages. If we know the C language, then we can easily learn other programming languages. C language was developed by the great computer scientist Dennis Ritchie at the Bell Laboratories. It contains some additional features that make it unique from other programming languages.

What is C++?

C++ is a special-purpose programming language developed by **Bjarne Stroustrup** at Bell Labs circa 1980. C++ language is very similar to C language, and it is so compatible with C that it can run 99% of C programs without changing any source of code though C++ is an object-oriented programming language, so it is safer and well-structured programming language than C.

Let's understand the differences between C and C++.



The following are the differences between C and C++:

- **Definition**
  C is a structural programming language, and it does not support classes and objects, while C++ is an object-oriented programming language that supports the concept of classes and objects.

- **Type of programming language**
  C supports the structural programming language where the code is checked line by line, while C++ is an object-oriented programming language that supports the concept of classes and objects.
- **Developer of the language**
  Dennis Ritchie developed C language at Bell Laboratories while Bjarne Stroustrup developed the C++ language at Bell Labs circa 1980.
- **Subset**
  C++ is a superset of C programming language. C++ can run 99% of C code but C language cannot run C++ code.
- **Type of approach**
  C follows the top-down approach, while C++ follows the bottom-up approach. The top-down approach breaks the main modules into tasks; these tasks are broken into sub-tasks, and so on. The bottom-down approach develops the lower level modules first and then the next level modules.
- **Security**
  In C, the data can be easily manipulated by the outsiders as it does not support the encapsulation and information hiding while C++ is a very secure language, i.e., no outsiders can manipulate its data as it supports both encapsulation and data hiding. In C language, functions and data are the free entities, and in C++ language, all the functions and data are encapsulated in the form of objects.
- **Function Overloading**
  Function overloading is a feature that allows you to have more than one function with the same name but varies in the parameters. C does not support the function overloading, while C++ supports the function overloading.
- **Function Overriding**
  Function overriding is a feature that provides the specific implementation to the function, which is already defined in the base class. C does not support the function overriding, while C++ supports the function overriding.
- **Reference variables**
  C does not support the reference variables, while C++ supports the reference variables.
- **Keywords**
  C contains 32 keywords, and C++ supports 52 keywords.
- **Namespace feature**
  A namespace is a feature that groups the entities like classes, objects, and functions under some specific name. C does not contain the namespace feature, while C++ supports the namespace feature that avoids the name collisions.
- **Exception handling**
  C does not provide direct support to the exception handling; it needs to use functions that

support exception handling. C++ provides direct support to exception handling by using a try-catch block.

- **Input/Output functions**
  In C, scanf and printf functions are used for input and output operations, respectively, while in C++, cin and cout are used for input and output operations, respectively.
- **Memory allocation and de-allocation**
  C supports calloc() and malloc() functions for the memory allocation, and free() function for the memory de-allocation. C++ supports a new operator for the memory allocation and delete operator for the memory de-allocation.
- **Inheritance**
  Inheritance is a feature that allows the child class to reuse the properties of the parent class. C language does not support the inheritance while C++ supports the inheritance.
- **Header file**
  C program uses **<stdio.h>** header file while C++ program uses **<iostream.h>** header file.

Let's summarize the above differences in a tabular form.

No.	C	C++
1)	C follows the **procedural style programming.**	C++ is multi-paradigm. It supports both **procedural and object oriented.**
2)	Data is less secured in C.	In C++, you can use modifiers for class members to make it inaccessible for outside users.
3)	C follows the **top-down approach.**	C++ follows the **bottom-up approach.**
4)	C does not support function overloading.	C++ supports function overloading.
5)	In C, you can't use functions in structure.	In C++, you can use functions in structure.
6)	C does not support reference variables.	C++ supports reference variables.
7)	In C, **scanf() and printf()** are mainly used for input/output.	C++ mainly uses stream **cin and cout** to perform input and output operations.
8)	Operator overloading is not possible in C.	Operator overloading is possible in C++.
9)	C programs are divided into **procedures and modules**	C++ programs are divided into **functions and classes.**

10)	C does not provide the feature of namespace.	C++ supports the feature of namespace.
11)	Exception handling is not easy in C. It has to perform using other functions.	C++ provides exception handling using Try and Catch block.
12)	C does not support the inheritance.	C++ supports inheritance.

← Prev Next →

C++ history

History of C++ language is interesting to know. Here we are going to discuss brief history of C++ language.

C++ programming language was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

Bjarne Stroustrup is known as the **founder of C++ language.**

It was develop for adding a feature of **OOP (Object Oriented Programming)** in C without significantly changing the C component.

C++ programming is "relative" (called a superset) of C, it means any valid C program is also a valid C++ program.

Let's see the programming languages that were developed before C++ language.

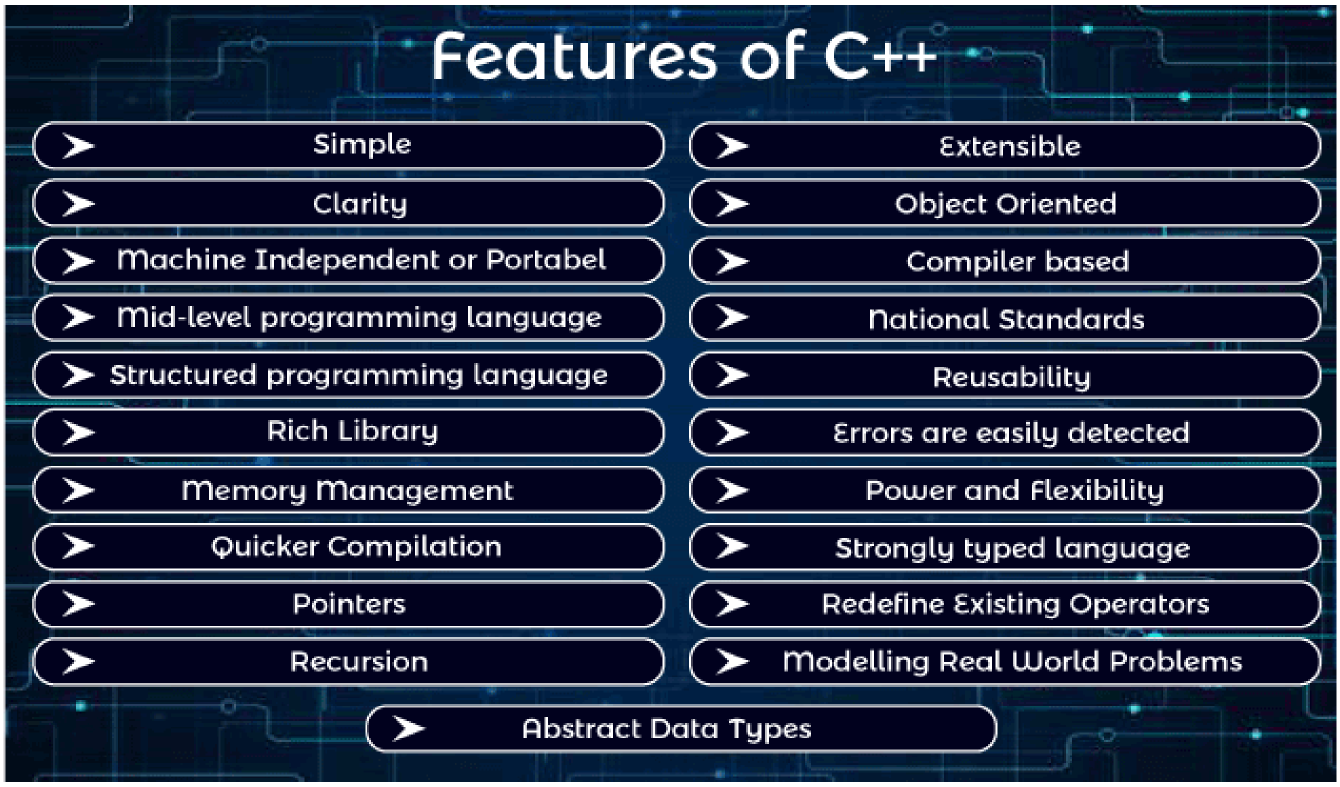
Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
C++	1980	Bjarne Stroustrup

← Prev

Next →

C++ Features

C++ is a widely used programming language.



It provides a lot of features that are given below.

1. Simple
2. Abstract Data types
3. Machine Independent or Portable
4. Mid-level programming language
5. Structured programming language
6. Rich Library
7. Memory Management
8. Quicker Compilation
9. Pointers
10. Recursion
11. Extensible
12. Object-Oriented
13. Compiler based
14. Reusability

15. National Standards
16. Errors are easily detected
17. Power and Flexibility
18. Strongly typed language
19. Redefine Existing Operators
20. Modeling Real-World Problems
21. Clarity

1) Simple

C++ is a simple language because it provides a structured approach (to break the problem into parts), a rich set of library functions, data types, etc.

2) Abstract Data types

In C++, complex data types called Abstract Data Types (ADT) can be created using classes.

3) Portable

C++ is a portable language and programs made in it can be run on different machines.

4) Mid-level / Intermediate programming language

C++ includes both low-level programming and high-level language so it is known as a mid-level and intermediate programming language. It is used to develop system applications such as kernel, driver, etc.

5) Structured programming language

C++ is a structured programming language. In this we can divide the program into several parts using functions.

6) Rich Library

C++ provides a lot of inbuilt functions that make the development fast. Following are the libraries used in C++ programming are:

- <iostream>
- <cmath>
- <cstdlib>

- <fstream>

7) Memory Management

C++ provides very efficient management techniques. The various memory management operators help save the memory and improve the program's efficiency. These operators allocate and deallocate memory at run time. Some common memory management operators available C++ are new, delete etc.

8) Quicker Compilation

C++ programs tend to be compact and run quickly. Hence the compilation and execution time of the C++ language is fast.

9) Pointer

C++ provides the feature of pointers. We can use pointers for memory, structures, functions, array, etc. We can directly interact with the memory by using the pointers.

10) Recursion

In C++, we can call the function within the function. It provides code reusability for every function.

11) Extensible

C++ programs can easily be extended as it is very easy to add new features into the existing program.

12) Object-Oriented

In C++, object-oriented concepts like data hiding, encapsulation, and data abstraction can easily be implemented using keyword class, private, public, and protected access specifiers. Object-oriented makes development and maintenance easier.

13) Compiler based

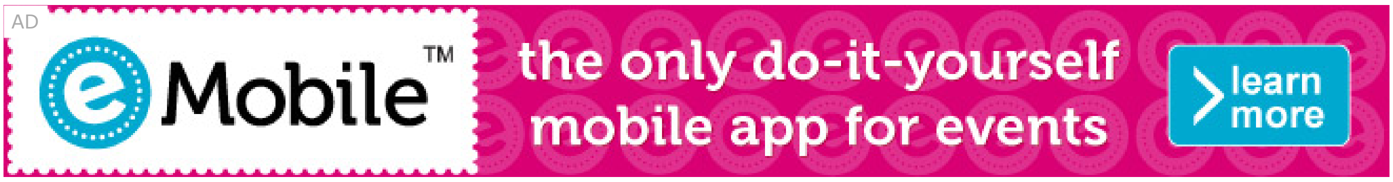
C++ is a compiler-based programming language, which means no C++ program can be executed without compilation. C++ compiler is easily available, and it requires very little space for storage. First, we need to compile our program using a compiler, and then we can execute our program.

14) Reusability

With the use of inheritance of functions programs written in C++ can be reused in any other program of C++. You can save program parts into library files and invoke them in your next programming projects simply by including the library files. New programs can be developed in lesser time as the existing code can be reused. It is also possible to define several functions with same name that perform different task. For Example: abs () is used to calculate the absolute value of integer, float and long integer.

15) National Standards

C++ has national standards such as ANSI.



16) Errors are easily detected

It is easier to maintain a C++ programs as errors can be easily located and rectified. It also provides a feature called exception handling to support error handling in your program.

17) Power and Flexibility

C++ is a powerful and flexible language because of most of the powerful flexible and modern UNIX operating system is written in C++. Many compilers and interpreters for other languages such as FORTRAN, PERL, Python, PASCAL, BASIC, LISP, etc., have been written in C++. C++ programs have been used for solving physics and engineering problems and even for animated special effects for movies.

18) Strongly typed language

The list of arguments of every function call is typed checked during compilation. If there is a type mismatch between actual and formal arguments, implicit conversion is applied if possible. A compile-time occurs if an implicit conversion is not possible or if the number of arguments is incorrect.

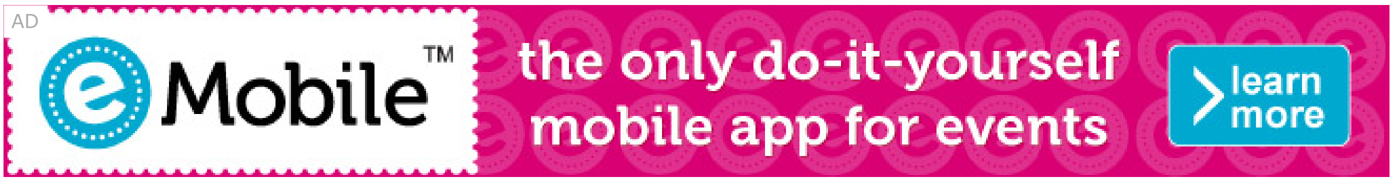
19) Redefine Existing Operators

C++ allows the programmer to redefine the meaning of existing operators such as +, -. **For Example,** The "+" operator can be used for adding two numbers and concatenating two strings.

20) Modelling real-world problems

The programs written in C++ are well suited for real-world modeling problems as close as possible to the user perspective.

AD



21) Clarity

The keywords and library functions used in C++ resemble common English words.

C++ Program

Before starting the abcd of C++ language, you need to learn how to write, compile and run the first C++ program.

To write the first C++ program, open the C++ console and write the following code:

```
#include <iostream.h>
#include<conio.h>
void main() {
   clrscr();
   cout << "Welcome to C++ Programming.";
   getch();
}
```

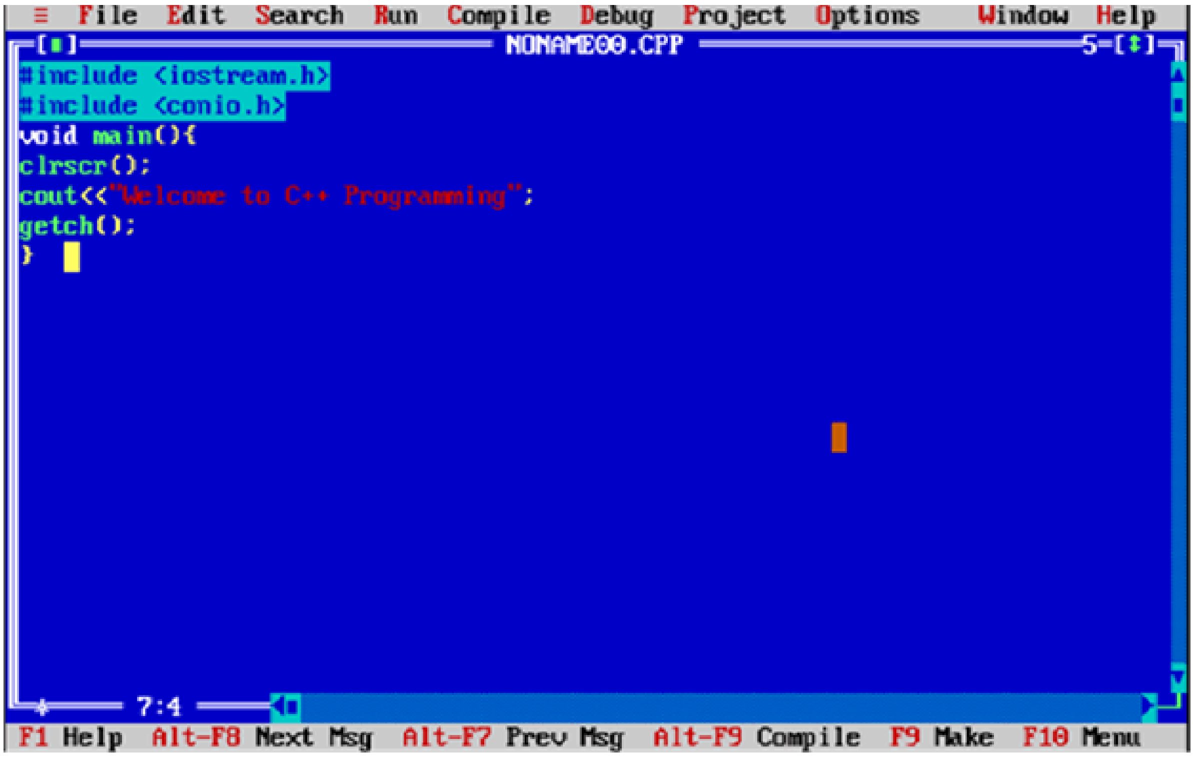
#include<iostream.h> includes the **standard input output** library functions. It provides **cin** and **cout** methods for reading from input and writing to output respectively.

#include <conio.h> includes the **console input output** library functions. The getch() function is defined in conio.h file.

void main() The **main() function is the entry point of every program** in C++ language. The void keyword specifies that it returns no value.

cout << "Welcome to C++ Programming." is **used to print the data "Welcome to C++ Programming."** on the console.

getch() The getch() function **asks for a single character**. Until you press any key, it blocks the screen.


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] NONAME00.CPP 5-[↕]
7:4
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include <iostream.h>
#include <conio.h>
void main(){
clrscr();
cout<<"Welcome to C++ Programming";
getch();
}
```

How to compile and run the C++ program

There are 2 ways to compile and run the C++ program, by menu and by shortcut.

By menu

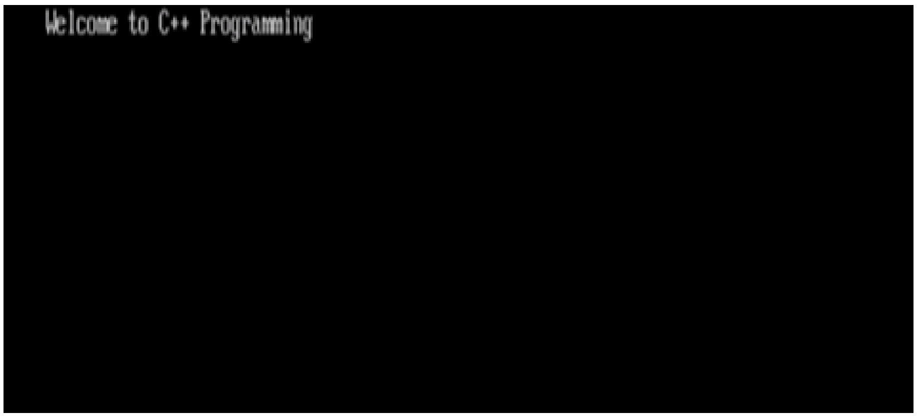
Now **click on the compile menu then compile sub menu** to compile the c++ program.

Then **click on the run menu then run sub menu** to run the c++ program.

By shortcut

Or, press ctrl+f9 keys compile and run the program directly.

You will see the following output on user screen.



```
Welcome to C++ Programming
```

You can view the user screen any time by pressing the **alt+f5** keys.

Now **press Esc** to return to the turbo c++ console.

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

C++ Basic Input/Output

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

I/O Library Header Files

Let us see the common header files used in C++ programming are:

Header File	Function and Description
<iostream>	It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively.
<iomanip>	It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.**
<fstream>	It is used to declare services for user-controlled file processing.

Standard output stream (cout)

The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

Let's see the simple example of standard output stream (cout):

```
#include <iostream>
using namespace std;
int main( ) {
   char ary[] = "Welcome to C++ tutorial";
   cout << "Value of ary is: " << ary << endl;
}
```

Output:

```
Value of ary is: Welcome to C++ tutorial
```

Standard input stream (cin)

The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Let's see the simple example of standard input stream (cin):

```
#include <iostream>
using namespace std;
int main( ) {
  int age;
   cout << "Enter your age: ";
   cin >> age;
   cout << "Your age is: " << age << endl;
}
```

Output:

```
Enter your age: 22
Your age is: 22
```

AD

Standard end line (endl)

The **endl** is a predefined object of **ostream** class. It is used to insert a new line characters and flushes the stream.

Let's see the simple example of standard end line (endl):

```
#include <iostream>
using namespace std;
int main( ) {
cout << "C++ Tutorial";
cout << " Javatpoint"<<endl;
cout << "End of line"<<endl;
}
```

Output:

```
C++ Tutorial Javatpoint
End of line
```

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

C++ Variable

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

```
type variable_list;
```

The example of declaring variable is given below:

```
int x;
float y;
char z;
```

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

```
int x=5,b=10;  //declaring 2 variable of integer type
float f=30.8;
char c='A';
```

Rules for defining variables

A variable can have alphabets, digits and underscore.

A variable name can start with alphabet and underscore only. It can't start with digit.

No white space is allowed within variable name.

A variable name must not be any reserved word or keyword e.g. char, float etc.

Valid variable names:

```
int a;
int _ab;
int a30;
```

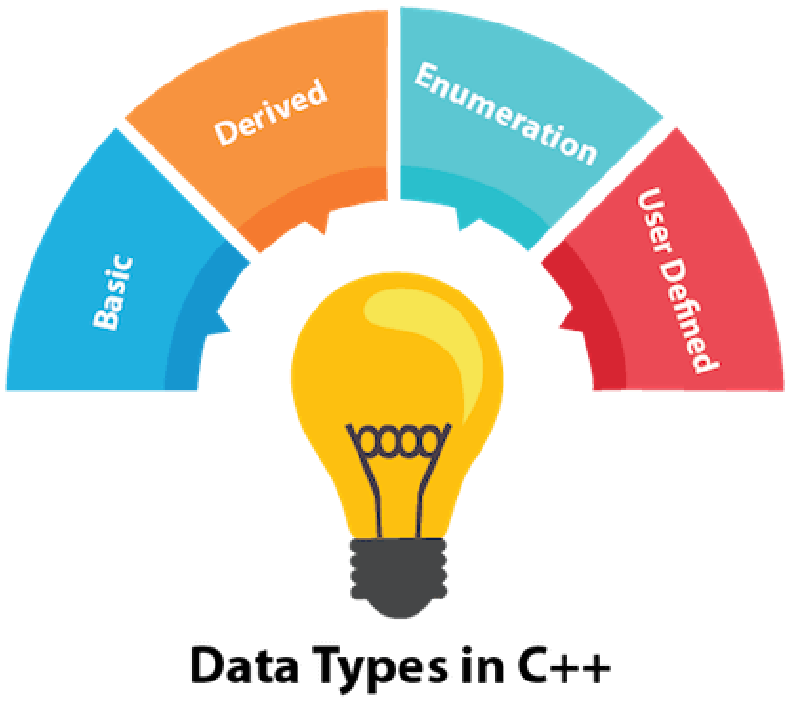
Invalid variable names:

```
int 4;
int x y;
int double;
```

← Prev Next →

C++ Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.



There are 4 types of data types in C++ language.

Types	Data Types
Basic Data Type	int, char, float, double, etc
Derived Data Type	array, pointer, etc
Enumeration Data Type	enum
User Defined Data Type	structure

Basic Data Types

The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32 or 64 bit operating system.

Let's see the basic data types. It size is given according to 32 bit OS.

Data Types	Memory Size	Range
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 127
short	2 byte	-32,768 to 32,767
signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 32,767
int	2 byte	-32,768 to 32,767
signed int	2 byte	-32,768 to 32,767
unsigned int	2 byte	0 to 32,767
short int	2 byte	-32,768 to 32,767
signed short int	2 byte	-32,768 to 32,767
unsigned short int	2 byte	0 to 32,767
long int	4 byte	
signed long int	4 byte	
unsigned long int	4 byte	
float	4 byte	
double	8 byte	
long double	10 byte	

← Prev Next →

C++ Keywords

A keyword is a reserved word. You cannot use it as a variable name, constant name etc. **A list of 32 Keywords in C++ Language which are also available in C language are given below.**

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

A list of 30 Keywords in C++ Language which are not available in C language are given below.

asm	dynamic_cast	namespace	reinterpret_cast	bool
explicit	new	static_cast	false	catch
operator	template	friend	private	class
this	inline	public	throw	const_cast
delete	mutable	protected	true	try
typeid	typename	using	virtual	wchar_t

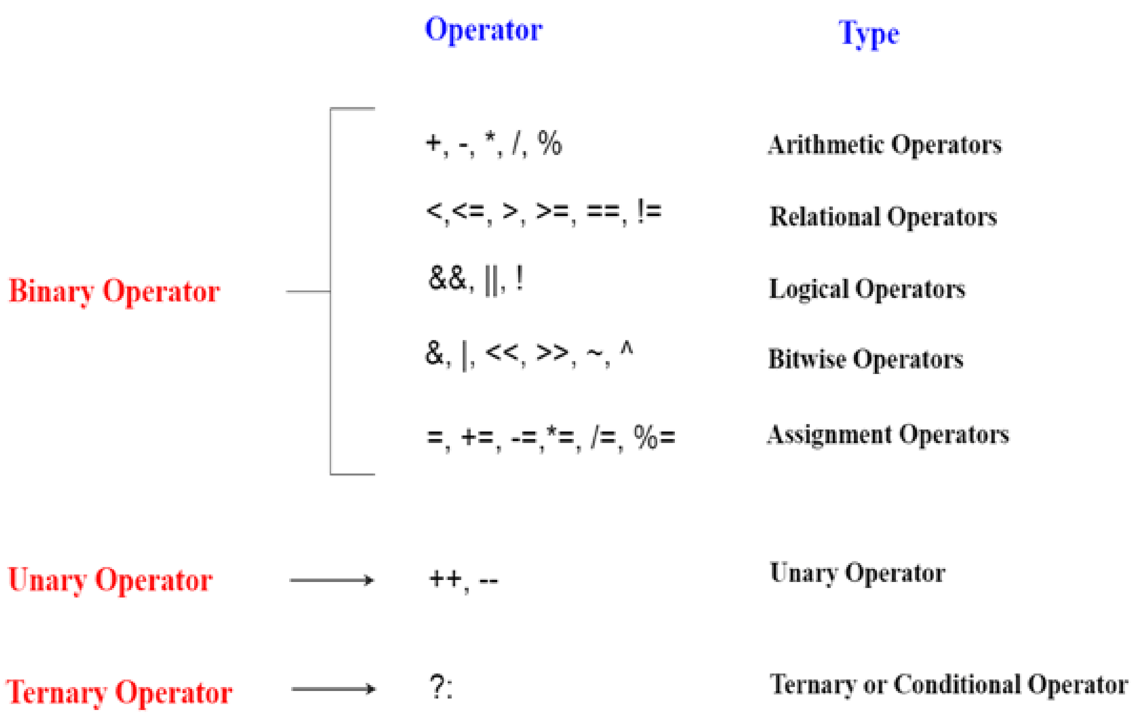
← Prev Next →

C++ Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operator
- Unary operator
- Ternary or Conditional Operator
- Misc Operator



	Operator	Type
Binary Operator	+, -, *, /, %	Arithmetic Operators
	<,<=, >, >=, ==, !=	Relational Operators
	&&, ||, !	Logical Operators
	&, |, <<, >>, ~, ^	Bitwise Operators
	=, +=, -=,*=, /=, %=	Assignment Operators
Unary Operator	++, --	Unary Operator
Ternary Operator	?:	Ternary or Conditional Operator

Precedence of Operators in C++

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operators direction to be evaluated, it may be left to right or right to left.

Let's understand the precedence by the example given below:

```
int data=5+10*10;
```

The "data" variable will contain 105 because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C++ operators is given below:

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Right to left
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=/td>	Right to left
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Right to left
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

← Prev Next →

C++ Identifiers

C++ identifiers in a program are used to refer to the name of the variables, functions, arrays, or other user-defined data types created by the programmer. They are the basic requirement of any language. Every language has its own rules for naming the identifiers.

In short, we can say that the C++ identifiers represent the essential elements in a program which are given below:

- **Constants**
- **Variables**
- **Functions**
- **Labels**
- **Defined data types**

Some naming rules are common in both C and C++. They are as follows:

- Only alphabetic characters, digits, and underscores are allowed.
- The identifier name cannot start with a digit, i.e., the first letter should be alphabetical. After the first letter, we can use letters, digits, or underscores.
- In C++, uppercase and lowercase letters are distinct. Therefore, we can say that C++ identifiers are case-sensitive.
- A declared keyword cannot be used as a variable name.

For example, suppose we have two identifiers, named as 'FirstName', and 'Firstname'. Both the identifiers will be different as the letter 'N' in the first case in uppercase while lowercase in second. Therefore, it proves that identifiers are case-sensitive.

Valid Identifiers

The following are the examples of valid identifiers are:

```
Result
Test2
_sum
```

```
power
```

Invalid Identifiers

The following are the examples of invalid identifiers:

```
Sum-1   // containing special character '-'.
2data   // the first letter is a digit.
break   // use of a keyword.
```

Note: Identifiers cannot be used as the keywords. It may not conflict with the keywords, but it is highly recommended that the keywords should not be used as the identifier name. You should always use a consistent way to name the identifiers so that your code will be more readable and maintainable.

The major difference between C and C++ is the limit on the length of the name of the variable. ANSI C considers only the first 32 characters in a name while ANSI C++ imposes no limit on the length of the name.

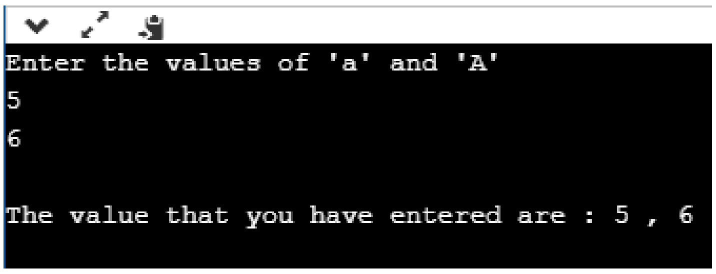
Constants are the identifiers that refer to the fixed value, which do not change during the execution of a program. Both C and C++ support various kinds of literal constants, and they do have any memory location. For example, 123, 12.34, 037, 0X2, etc. are the literal constants.

Let's look at a simple example to understand the concept of identifiers.

```
#include <iostream>
using namespace std;
int main()
{
   int a;
   int A;
   cout<<"Enter the values of 'a' and 'A'";
   cin>>a;
   cin>>A;
   cout<<"\nThe values that you have entered are : "<<a<<" , "<<A;
   return 0;
}
```

In the above code, we declare two variables 'a' and 'A'. Both the letters are same but they will behave as different identifiers. As we know that the identifiers are the case-sensitive so both the identifiers will have different memory locations.

Output



```
Enter the values of 'a' and 'A'
5
6

The value that you have entered are : 5 , 6
```

What are the keywords?

Keywords are the reserved words that have a special meaning to the compiler. They are reserved for a special purpose, which cannot be used as the identifiers. For example, 'for', 'break', 'while', 'if', 'else', etc. are the predefined words where predefined words are those words whose meaning is already known by the compiler. Whereas, the identifiers are the names which are defined by the programmer to the program elements such as variables, functions, arrays, objects, classes.

Differences between Identifiers and Keywords

The following is the list of differences between identifiers and keywords:

Identifiers	Keywords
Identifiers are the names defined by the programmer to the basic elements of a program.	Keywords are the reserved words whose meaning is known by the compiler.
It is used to identify the name of the variable.	It is used to specify the type of entity.
It can consist of letters, digits, and underscore.	It contains only letters.
It can use both lowercase and uppercase letters.	It uses only lowercase letters.
No special character can be used except the underscore.	It cannot contain any special character.
The starting letter of identifiers can be lowercase, uppercase or underscore.	It can be started only with the lowercase letter.
It can be classified as internal and external identifiers.	It cannot be further classified.

Examples are test, result, sum, power, etc.	Examples are 'for', 'if', 'else', 'break', etc.

← Prev Next →

AD

Youtube For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk tutorial Splunk	SPSS tutorial SPSS	Swagger tutorial Swagger	T-SQL tutorial Transact-SQL
Tumblr tutorial Tumblr	React tutorial ReactJS	Regex tutorial Regex	Reinforcement learning tutorial

C++ Expression

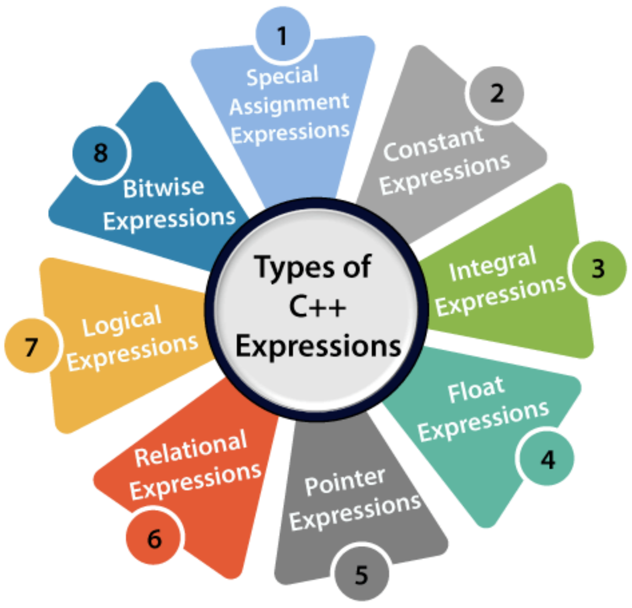
C++ expression consists of operators, constants, and variables which are arranged according to the rules of the language. It can also contain function calls which return values. An expression can consist of one or more operands, zero or more operators to compute a value. Every expression produces some value which is assigned to the variable with the help of an assignment operator.

Examples of C++ expression:

```
(a+b) - c
(x/y) -z
4a2 - 5b +c
(a+b) * (x+y)
```

An expression can be of following types:

- Constant expressions
- Integral expressions
- Float expressions
- Pointer expressions
- Relational expressions
- Logical expressions
- Bitwise expressions
- Special assignment expressions



If the expression is a combination of the above expressions, such expressions are known as compound expressions.

Constant expressions

A constant expression is an expression that consists of only constant values. It is an expression whose value is determined at the compile-time but evaluated at the run-time. It can be composed of integer, character, floating-point, and enumeration constants.

Constants are used in the following situations:

- It is used in the subscript declarator to describe the array bound.
- It is used after the case keyword in the switch statement.
- It is used as a numeric value in an **enum**

- It specifies a bit-field width.
- It is used in the pre-processor **#if**

In the above scenarios, the constant expression can have integer, character, and enumeration constants. We can use the static and extern keyword with the constants to define the function-scope.

The following table shows the expression containing constant value:

Expression containing constant	Constant value
x = (2/3) * 4	(2/3) * 4
extern int y = 67	67
int z = 43	43
static int a = 56	56

Let's see a simple program containing constant expression:

```
#include <iostream>
using namespace std;
int main()
{
    int x;        // variable declaration.
    x=(3/2) + 2;  // constant expression
    cout<<"Value of x is : "<<x;  // displaying the value of x.
    return 0;
}
```

In the above code, we have first declared the 'x' variable of integer type. After declaration, we assign the simple constant expression to the 'x' variable.

Output

```
Value of x is : 3
```

Integral Expressions

An integer expression is an expression that produces the integer value as output after performing all the explicit and implicit conversions.

Following are the examples of integral expression:

```
(x * y) -5
x + int(9.0)
where x and y are the integers.
```

Let's see a simple example of integral expression:

```
#include <iostream>
using namespace std;
int main()
{
    int x;  // variable declaration.
    int y;  // variable declaration
    int z;  // variable declaration
    cout<<"Enter the values of x and y";
    cin>>x>>y;
    z=x+y;
    cout<<"\n"<<"Value of z is :"<<z; //  displaying the value of z.
    return 0;
}
```

In the above code, we have declared three variables, i.e., x, y, and z. After declaration, we take the user input for the values of 'x' and 'y'. Then, we add the values of 'x' and 'y' and stores their result in 'z' variable.

Output

```
Enter the values of x and y
8
9
Value of z is :17
```

Let's see another example of integral expression.

```
#include <iostream>
using namespace std;
int main()
{

   int x;   // variable declaration
   int y=9;    // variable initialization
   x=y+int(10.0);    // integral expression
  cout<<"Value of x : "<<x;   // displaying the value of x.
  return 0;
}
```

In the above code, we declare two variables, i.e., x and y. We store the value of expression (y+int(10.0)) in a 'x' variable.

Output

```
Value of x : 19
```

Float Expressions

A float expression is an expression that produces floating-point value as output after performing all the explicit and implicit conversions.

The following are the examples of float expressions:

```
x+y
(x/10) + y
34.5
x+float(10)
```

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{


   float x=8.9;      // variable initialization
   float y=5.6;      // variable initialization
   float z;          // variable declaration
   z=x+y;
   std::cout <<"value of z is :"  << z<<std::endl;  // displaying the value of z.



    return 0;
}
```

Output

```
value of z is :14.5
```

Let's see another example of float expression.

```
#include <iostream>
using namespace std;
int main()
{
   float x=6.7;   // variable initialization
   float y;      // variable declaration
   y=x+float(10);   // float expression
   std::cout <<"value of y is :"  << y<<std::endl;  // displaying the value of y
   return 0;
}
```

In the above code, we have declared two variables, i.e., x and y. After declaration, we store the value of expression (x+float(10)) in variable 'y'.

Output

```
value of y is :16.7
```

Pointer Expressions

A pointer expression is an expression that produces address value as an output.

The following are the examples of pointer expression:

```
&x
ptr
ptr++
ptr-
```

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{

   int a[]={1,2,3,4,5};  // array initialization
   int *ptr;       // pointer declaration
   ptr=a;    // assigning base address of array to the pointer ptr
   ptr=ptr+1;   // incrementing the value of pointer
   std::cout <<"value of second element of an array : "  << *ptr<<std::endl;
   return 0;
}
```

In the above code, we declare the array and a pointer ptr. We assign the base address to the variable 'ptr'. After assigning the address, we increment the value of pointer 'ptr'. When pointer is incremented then 'ptr' will be pointing to the second element of the array.

Output

```
value of second element of an array : 2
```

Relational Expressions

A relational expression is an expression that produces a value of type bool, which can be either true or false. It is also known as a boolean expression. When arithmetic expressions are used on both sides of the relational operator, arithmetic expressions are evaluated first, and then their results are compared.

The following are the examples of the relational expression:

```
a>b
a-b >= x-y
a+b>80
```

Let's understand through an example

```
#include <iostream>
using namespace std;
int main()
{
    int a=45;    // variable declaration
    int b=78;    // variable declaration
    bool y= a>b;   // relational expression
    cout<<"Value of y is :"<<y;  // displaying the value of y.
    return 0;
}
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. After declaration, we have applied the relational operator between the variables to check whether 'a' is greater than 'b' or not.

Output

```
Value of y is :0
```

Let's see another example.

```
#include <iostream>
using namespace std;
int main()
{
 int a=4;     // variable declaration
 int b=5;     // variable declaration
 int x=3;     // variable declaration
 int y=6;    // variable declaration
 cout<<((a+b)>=(x+y));   // relational expression
 return 0;
}
```

In the above code, we have declared four variables, i.e., 'a', 'b', 'x' and 'y'. Then, we apply the relational operator (>=) between these variables.

Output

```
1
```

Logical Expressions

A logical expression is an expression that combines two or more relational expressions and produces a bool type value. The logical operators are '&&' and '||' that combines two or more relational expressions.

The following are some examples of logical expressions:

```
a>b && x>y
a>10 || b==5
```

Let's see a simple example of logical expression.

```
#include <iostream>
using namespace std;
int main()
{
 int a=2;
 int b=7;
 int c=4;
cout<<((a>b)||(a>c));
return 0;
}
```

Output

```
0
```

Bitwise Expressions

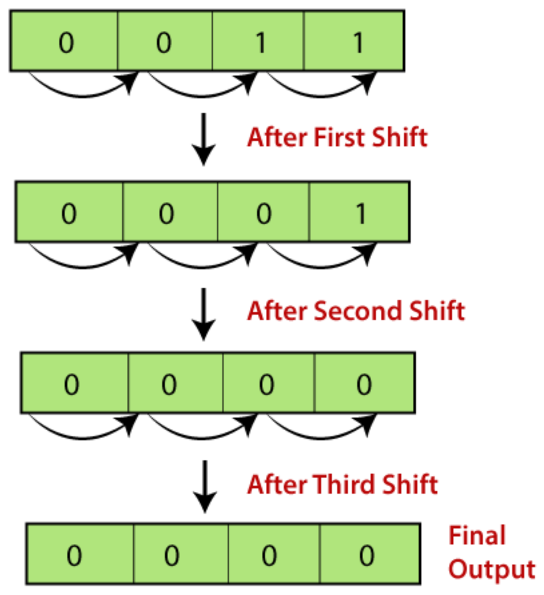
A bitwise expression is an expression which is used to manipulate the data at a bit level. They are basically used to shift the bits.

For example:

x=3

x>>3 // This statement means that we are shifting the three-bit position to the right.

In the above example, the value of 'x' is 3 and its binary value is 0011. We are shifting the value of 'x' by three-bit position to the right. Let's understand through the diagrammatic representation.



Let's see a simple example.

```
#include <iostream>
using namespace std;
int main()
{
 int x=5;   // variable declaration
std::cout << (x>>1) << std::endl;
return 0;
}
```

In the above code, we have declared a variable 'x'. After declaration, we applied the bitwise operator, i.e., right shift operator to shift one-bit position to right.

Output

```
2
```

Let's look at another example.

```
#include <iostream>
using namespace std;
int main()
{
 int x=7;   // variable declaration
std::cout << (x<<3) << std::endl;
return 0;
}
```

In the above code, we have declared a variable 'x'. After declaration, we applied the left shift operator to variable 'x' to shift the three-bit position to the left.

Output

```
56
```

Special Assignment Expressions

Special assignment expressions are the expressions which can be further classified depending upon the value assigned to the variable.

- **Chained Assignment**

Chained assignment expression is an expression in which the same value is assigned to more than one variable by using single statement.

For example:

```
a=b=20
 or
(a=b) = 20
```

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()

 int a;   // variable declaration
 int b;   // variable declaration
 a=b=80;  // chained assignment
 std::cout <<"Values of 'a' and 'b' are : " <<a<<","<<b<< std::endl;
 return 0;
}
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. Then, we have assigned the same value to both the variables using chained assignment expression.

Output

```
Values of 'a' and 'b' are : 80,80
```

> Note: Using chained assignment expression, the value cannot be assigned to the variable at the time of declaration. For example, int a=b=c=90 is an invalid statement.

- **Embedded Assignment Expression**

An embedded assignment expression is an assignment expression in which assignment expression is enclosed within another assignment expression.

Let's understand through an example.

```
#include <iostream>
```

```
using namespace std;
int main()
{
 int a;  // variable declaration
 int b;  // variable declaration
 a=10+(b=90);  // embedded assignment expression
 std::cout <<"Values of 'a' is " <<a<< std::endl;
 return 0;
}
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. Then, we applied embedded assignment expression (a=10+(b=90)).

Output

```
Values of 'a' is 100
```

- **Compound Assignment**

A compound assignment expression is an expression which is a combination of an assignment operator and binary operator.

For example,

```
a+=10;
```

In the above statement, 'a' is a variable and '+=' is a compound statement.

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{
  int a=10;   // variable declaration
  a+=10;    // compound assignment
  std::cout << "Value of a is :" <<a<< std::endl; // displaying the value of a.
  return 0;
}
```

In the above code, we have declared a variable 'a' and assigns 10 value to this variable. Then, we applied compound assignment operator (+=) to 'a' variable, i.e., a+=10 which is equal to (a=a+10). This statement increments the value of 'a' by 10.

Output

```
Value of a is :20
```

← Prev Next →