

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.

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

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

1. `#include <iostream>`
2. `using namespace std;`

```
3. int main() {  
4.     cout << "Hello C++ Programming";  
5.     return 0;  
6. }
```

What is C?

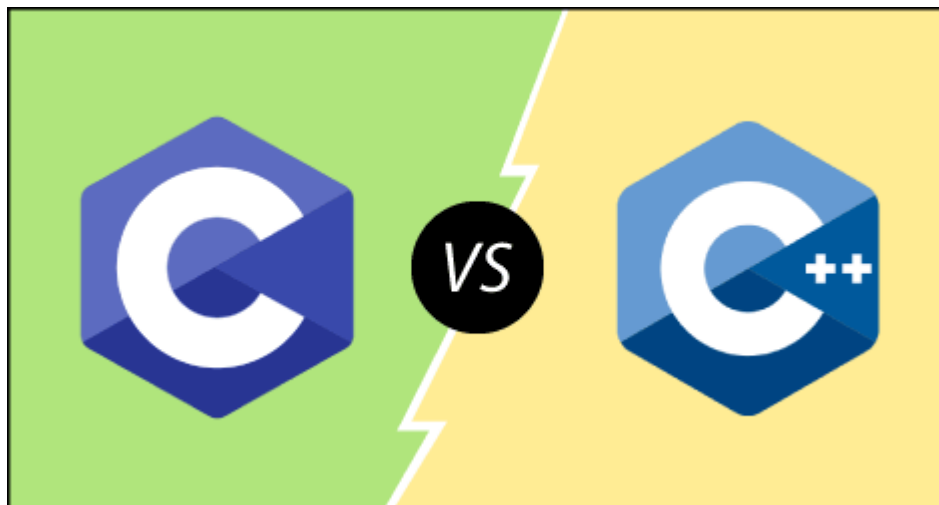
C is a structural or procedural oriented programming language which is machine-independent and extensively used in various applications.

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.

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.

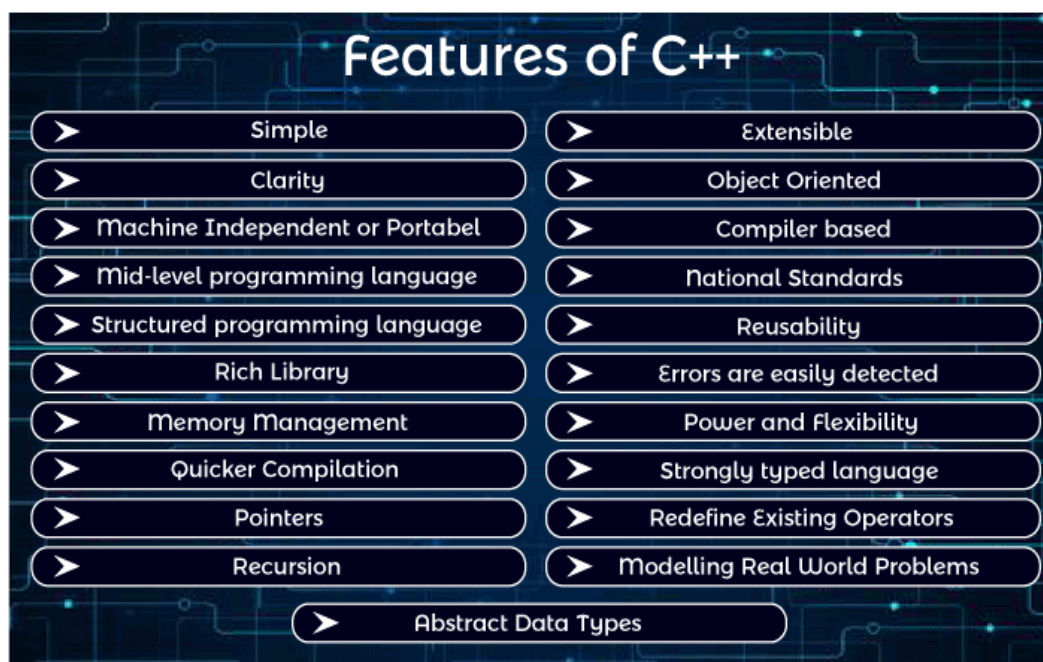
Let's see the programming languages that were developed before C++ language.



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

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.

21) Clarity

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

Turbo C++ - Download & Installation

There are many compilers available for C++. You need to download any one. Here, we are going to use **Turbo C++**. It will work for both C and C++. To install the Turbo C++ software, you need to follow following steps.

1. Download Turbo C++
2. Create turboc directory inside c drive and extract the tc3.zip inside c:\turboc
3. Double click on install.exe file
4. Click on the tc application file located inside c:\TC\BIN to write the c program

1) Download Turbo C++ software

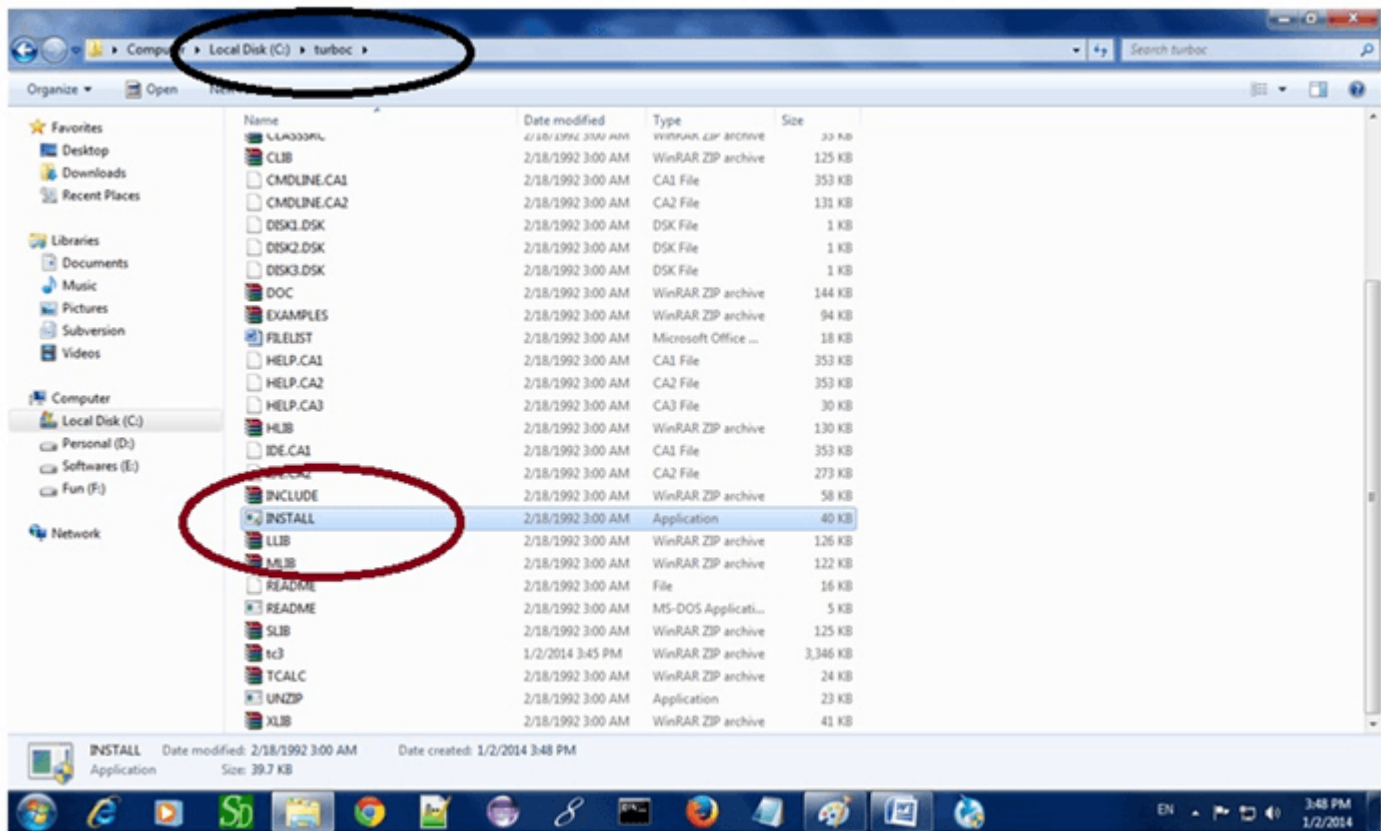
You can download turbo C++ from many sites. [download Turbo c++](#)

2) Create turboc directory in c drive and extract the tc3.zip

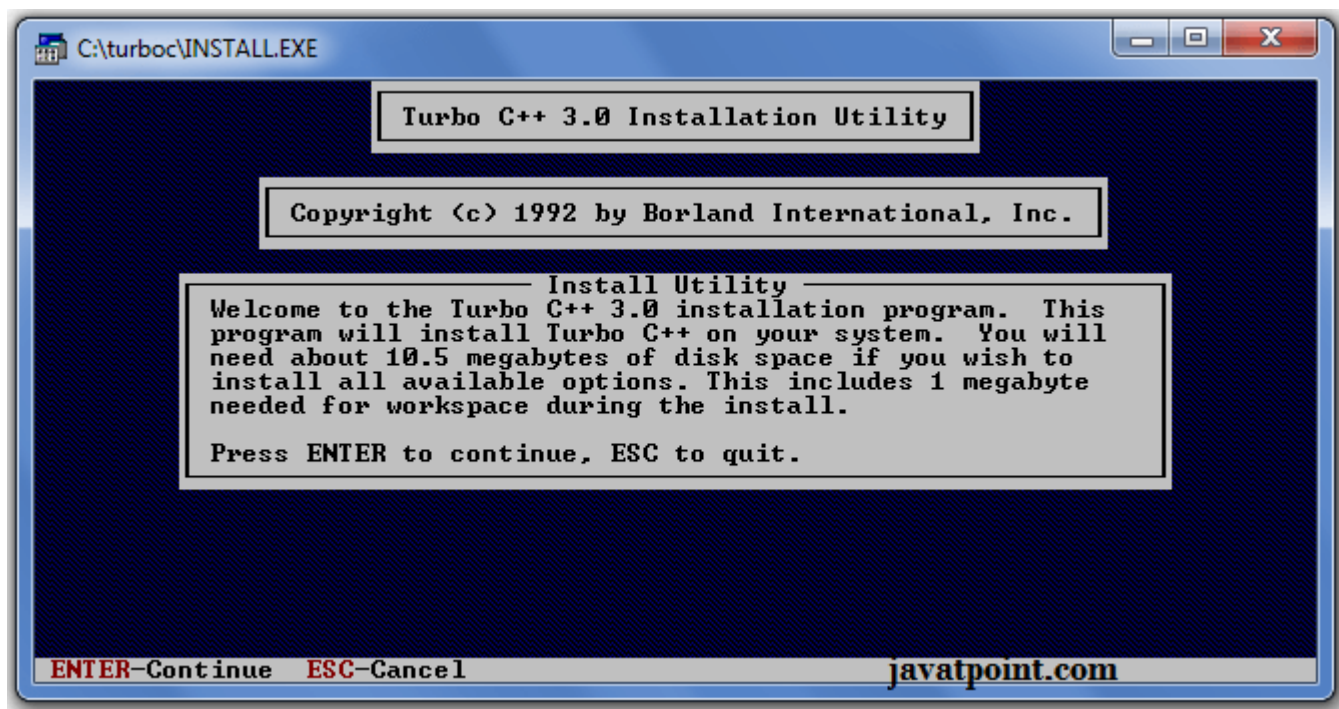
Now, you need to create a new directory turboc inside the c: drive. Now extract the tc3.zip file in c:\turboc directory.

3) Double click on the install.exe file and follow steps

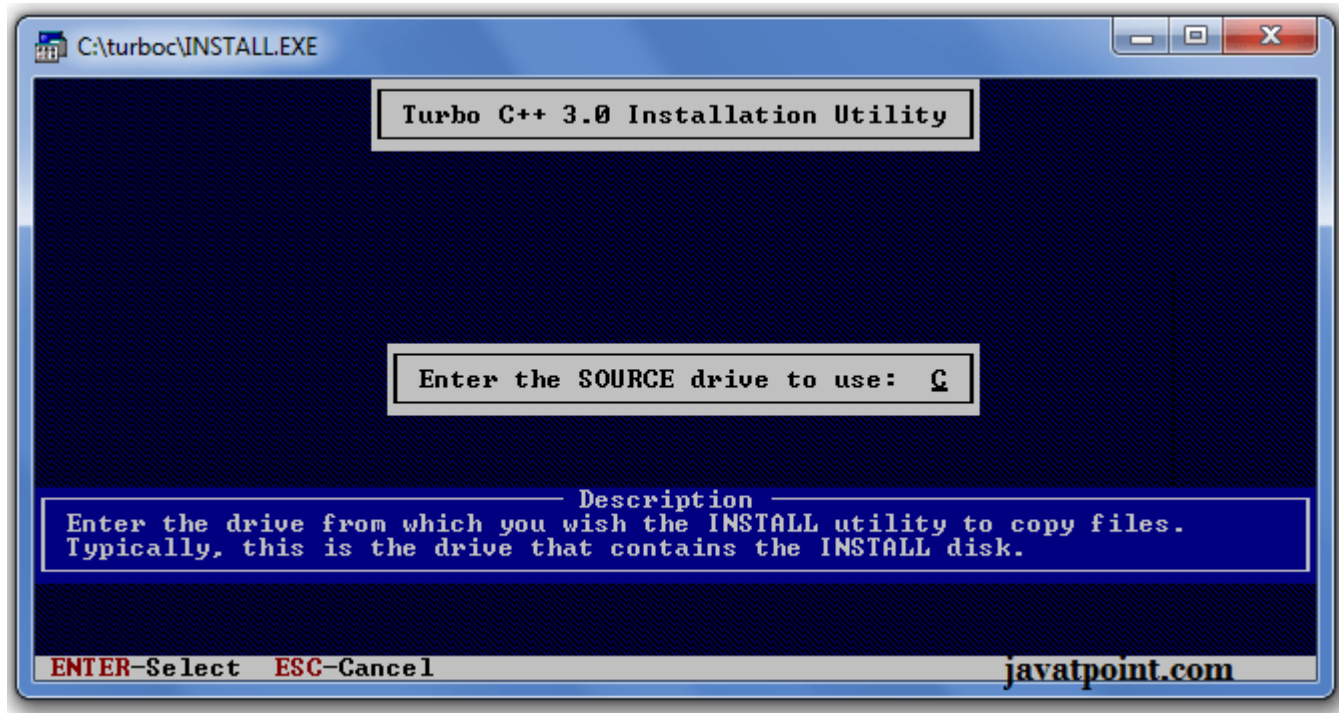
Now, click on the install icon located inside the c:\turboc



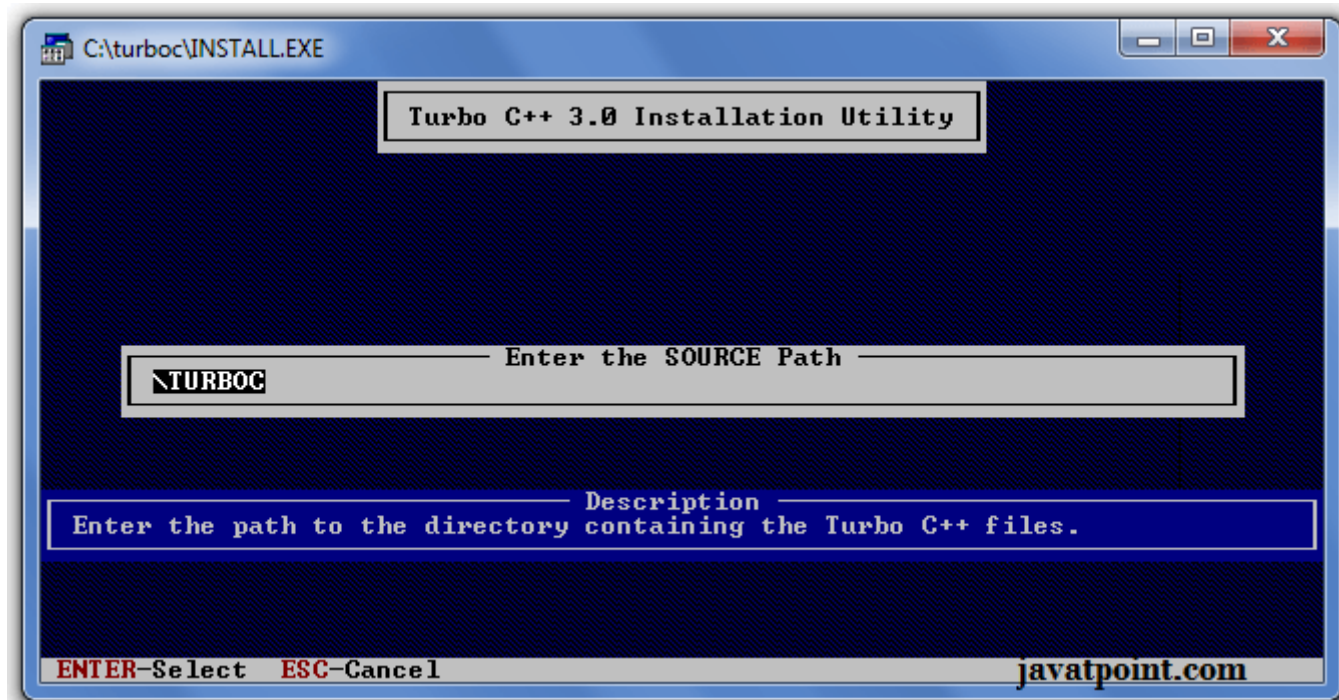
It will ask you to install c or not, press enter to install.



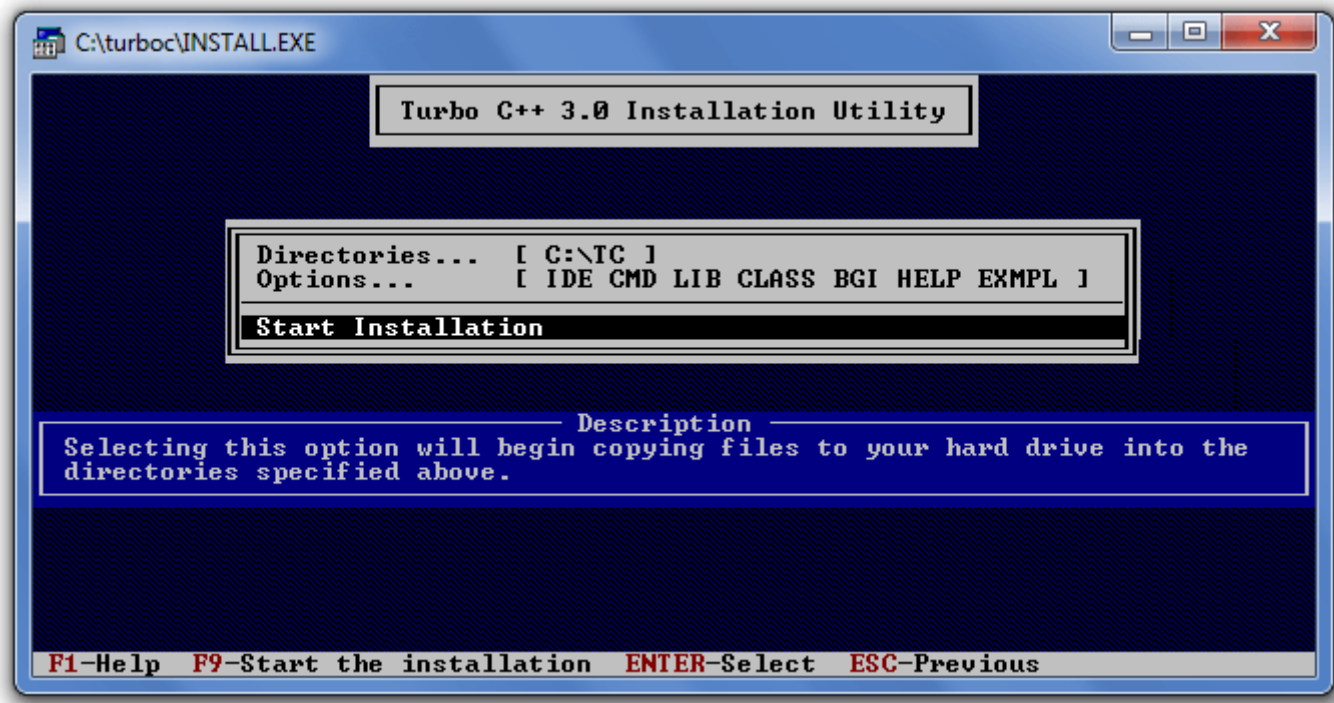
Change your drive to c, press c.



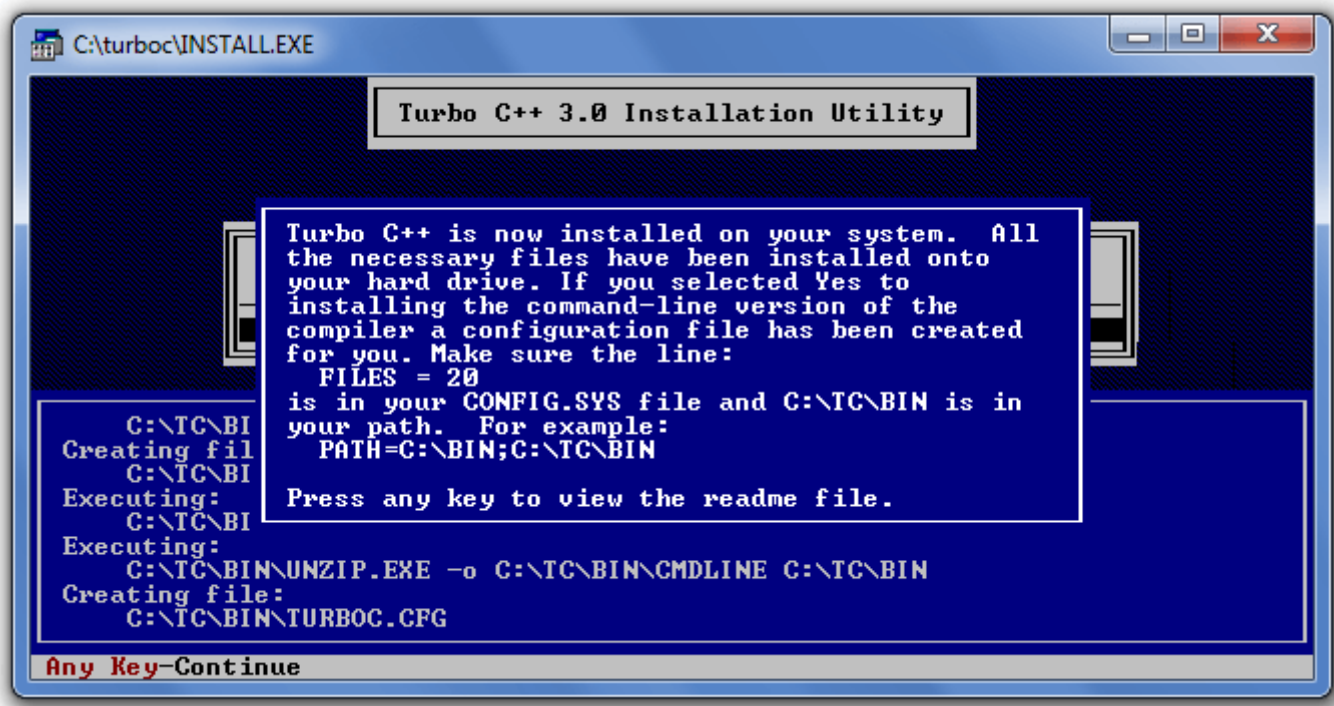
Press enter, it will look inside the c:\turboc directory for the required files.



Select Start installation by the down arrow key then press enter.

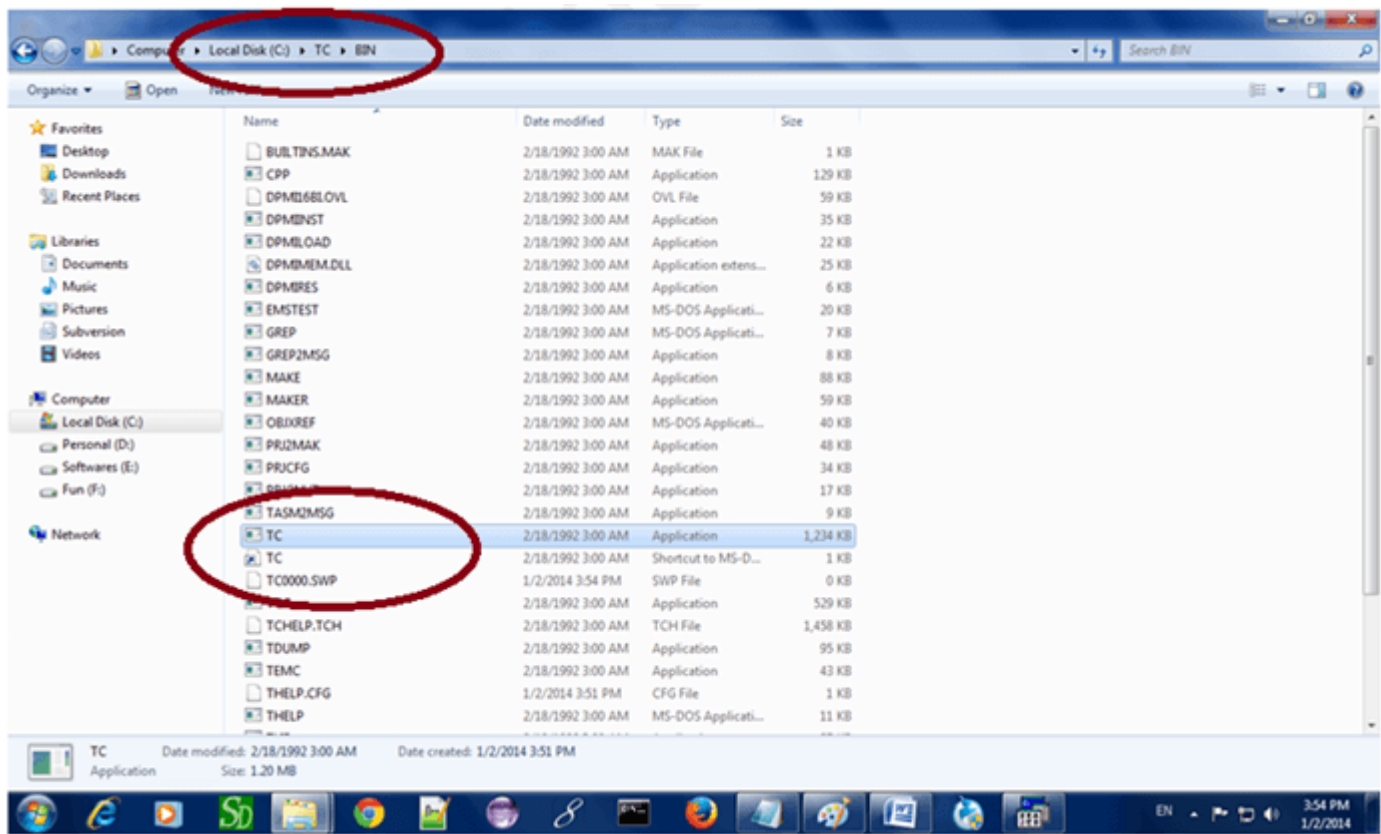


Now C is installed, press enter to read documentation or close the software.



4) Click on the tc application located inside c:\TC\BIN

Now double click on the tc icon located in c:\TC\BIN directory to write the c program.



In windows 7 or window 8, it will show a dialog block to ignore and close the application because full screen mode is not supported. Click on Ignore button.

Now it will showing following console.



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:

1. `#include <iostream.h>`
2. `#include<conio.h>`
3. `void main() {`
4. `clrscr();`
5. `cout << "Welcome to C++ Programming.";`
6. `getch();`
7. `}`

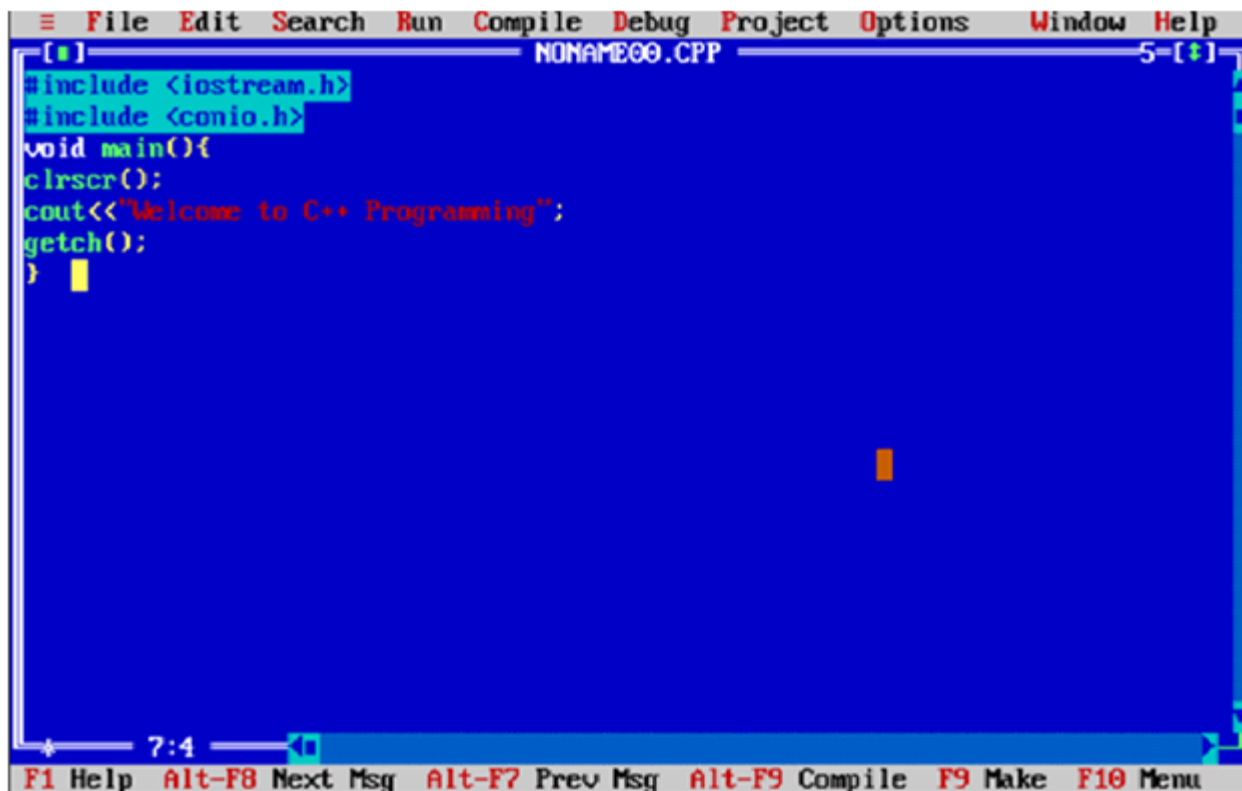
`#include<iostream.h>` includes the **standard input output** library functions. It provides **cin** and **cout** methods for reading from input and writing to output respectively.

`#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
#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.



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):

```
1. #include <iostream>
2. using namespace std;
3. int main( ) {
4.     char ary[] = "Welcome to C++ tutorial";
5.     cout << "Value of ary is: " << ary << endl;
6. }
```

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):

```
1. #include <iostream>
2. using namespace std;
3. int main( ) {
4.     int age;
5.     cout << "Enter your age: ";
6.     cin >> age;
7.     cout << "Your age is: " << age << endl;
8. }
```

Output:

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

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):

```
1. #include <iostream>
```

```
2. using namespace std;
3. int main() {
4. cout << "C++ Tutorial";
5. cout << " Javatpoint"<<endl;
6. cout << "End of line"<<endl;
7. }
```

Output:

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

C++ Variable

Variables are the fundamental building blocks of data manipulation and storage in programming, acting as *dynamic containers* for data in the C++ programming language. A **variable** is more than just a memory label. It serves as a link between abstract ideas and concrete data storage, allowing programmers to deftly manipulate data.

With the help of C++ variables, developers may complete a wide range of jobs, from simple *arithmetic operations* to complex algorithmic designs. These programmable containers can take on a variety of shapes, such as *integers*, *floating-point numbers*, *characters*, and *user-defined structures*, each of which has a distinctive impact on the operation of the program.

Programmers follow a set of guidelines when generating variables, creating names that combine alphanumeric letters and underscores while avoiding reserved keywords. More than just placeholders, variables are what drive *user input*, *intermediary calculations*, and the dynamic interactions that shape the program environment.

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:

```
1. type variable_list;
```

The example of declaring variable is given below:

```
1. int x;
2. float y;
3. 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:

```
1. int x=5,b=10; //declaring 2 variable of integer type
2. float f=30.8;
3. 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:

1. `int a;`
2. `int _ab;`
3. `int a30;`

Invalid variable names:

1. `int 4;`
2. `int x y;`
3. `int double;`

Uses of C++ Variables:

There are several uses of variables in C++. Some main uses of C++ variables are as follows:

Important Ideas: Programming is fundamentally based on C++ variables, which allow for the *storing, manipulation*, and *interaction* of data inside a program.

Memory Storage: Variables are named *memory regions* that may hold values of different data kinds, ranging from characters and integers to more intricate user-defined structures.

Dynamic character: Programming that is responsive and dynamic is made possible by the ability to *assign, modify*, and *reuse* data through variables.

Data Types: The *several data types* that C++ provides, including *int, float, char*, and others, each define the sort of value that a variable may store.

Variable declaration: Use the syntax *type variable_name* to define a variable, containing its *type* and *name*.

Initialization: When a variable is declared, it can be given a value, such as *int age = 25*.

Rules and Naming: Variable names must begin with a letter or an *underscore*, avoid reserved *keywords*, and be composed of *letters, numbers*, and *underscores*.

Utilization and Manipulation: The functionality of a program is improved by variables' participation in *arithmetic, logical, and relational operations*.

Scope: Variables have a scope that specifies the areas of a program where they may be accessed and used.

Reusability and Modularity: Variables with appropriate names make code easier to *comprehend, encourage modularity*, and allow for code reuse.

Object-Oriented: In object-oriented programming, variables are essential because they contain data inside of classes and objects.

Memory Control: Incorrect usage of variables can result in memory leaks or inefficient allocation, therefore understanding those helps with memory management.

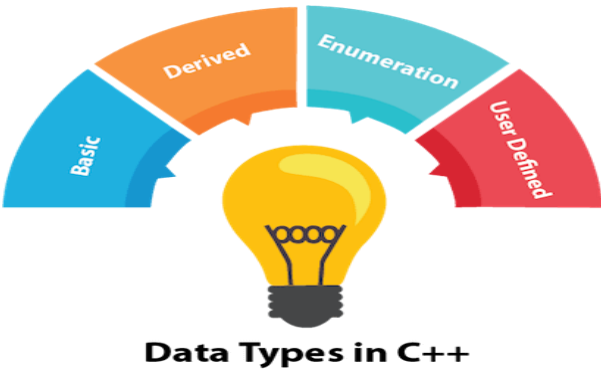
Applications in the Real World: Variables are used in a variety of applications, including *web applications*, *system programming*, and *scientific simulations*.

Debugging and upkeep: The proper use of variables reduces errors and enhances program quality while making debugging and code maintenance easier.

Interactivity: Variables are essential for interactive programs to capture user input and enable dynamic replies.

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	

The *nature* and size of variables in C++ are heavily influenced by *data types*, which also have an impact on *memory use* and the range of values that may be stored. Although the material given covers the fundamental data types, certain significant aspects and factors might improve your comprehension of C++ data types.

Types of Floating-Point Data:

C++ incorporates the idea of *scientific notation* for *encoding floating-point* literals in addition to *float*, *double*, and *long double*. In this system, exponentiation is denoted by the letter e or 'E'. For illustration:

1. `float scientificNotation = 3.0e5; // Represents 300000.0`

Fixed-Width Integer Types: C++11 added *fixed-width integer types* to ensure consistent behavior across various platforms, which contain a set number of bits. These types, whose names include *int8_t*, *uint16_t*, and *int32_t*, are specified in the *cstdint* header. Regardless of the underlying system, these types are particularly helpful when you want precise control over the size of numbers.

size of Operator: The *sizeof operator* is used to calculate a *data type* or variable's size (in bytes). For instance:

1. `int sizeOfInt = sizeof(int);`

Character and String Types: The C++ language uses the *char data type* to represent *characters*. *Wide characters* (*wchar_t*) for expanded character sets, and the *char16_t* and *char32_t types* for *Unicode characters* are also introduced by C++. The *std::string* class or character arrays (*char* or *wchar_t*) are used to represent strings.

References and Pointers: Despite the brief mention of derived *data types*, *pointers*, and *references* are crucial concepts in C++. A variable that stores the memory address of another variable is known as a *pointer*. It permits the allocation and modification of *dynamic memory*. On the other hand, *references* offer another method of accessing a variable by establishing an *alias*. *Pointers* and *references* are essential for complicated data structures and more sophisticated memory management.

Enumeration Data Type: An *enumeration* is a user-defined data type made up of named constants (*enumerators*), and it is specified by the *enum data type*. *Enumerations* are frequently used to increase the readability and maintainability of code by giving specified *data names* that make sense.

Structures and Classes as User-Defined Data Types: Although the *struct* was described in the example as a *user-defined data type*, C++ also introduces the concept of classes. Classes provide you with the ability to build *user-defined types* that are more complicated and have *member variables* and *related methods (functions)*. They serve as the foundation of C++'s object-oriented programming.

Bit Fields: C++ offers a method to declare how many bits should be used by each component of a structure. It is helpful when working with packed data structures or hardware registers.

Data Type Modifiers: C++ supports the usage of *data type modifiers* like *const*, *volatile*, and *mutable* to modify the behavior of variables. For example, *volatile* denotes that a *variable* can be altered outside, but *const* makes a variable *immutable*.

C++ Keywords

C++ *keywords* play a crucial role in defining the syntax and functioning of the language. They include reserved words with functions, such as specifying *data types*, managing *program flow*, and activating additional features. Understanding these terms is essential for good C++ programming and enables programmers to build *reliable* and *adaptable software*.

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

Keywords Available in Both C and C++:

Let's discuss these C++ keywords one by one.

auto: In C++, the auto keyword is mainly used for *type inference*. It enables the compiler to determine a variable's data type from its *initializer expression*.

break: By using the *break keyword*, the execution of a *loop* or *switch statement* is stopped.

case: It is used in a *switch statement* to provide several scenarios in which different code blocks should be run depending on the result of an expression.

char: The char keyword is used to declare character *variables* or *data types*.

const: *Constant values* or *pointers* that refer to constants are declared with the keyword *const*.

Continue: This keyword is used to go on to the next *iteration* of a loop without running the last remaining statements.

default: When no other case *matches*, it is used in a *switch statement* to provide a default case.

Do: This keyword initiates a *loop* that runs a series of statements as long as a certain condition is met.

double: *Double-precision floating-point* variables and data types are declared with the ***double keyword***.

else: It works in conjunction with the ***if statement*** to specify an alternative code snippet that is run if the condition is *false*.

enum: An *enumeration* is defined by the ***enum keyword***. It is a user-defined type that is made up of a collection of named integer constants.

extern: It is used to declare *variables* or *functions* that are declared in another source file.

float: The float keyword is used to declare *single-precision floating-point* variables or data types.

for: It begins a *loop* by introducing *initialization*, *condition*, and *update expressions*.

goto: The ***goto keyword*** is used to give an unconditional jump from the goto to a labeled statement within the same function.

If: This *operator* is used to run a block of code only if a certain condition is met.

int: The int keyword is used to declare the *integer variables* or *data types*.

long: The ***long keyword*** is utilized to declare the *integer variables* with a wider range than int.

register: The compiler is advised to put the variable in a *register* for quicker access.

return: This keyword is used to *end a function* and, if desired, to provide the caller a value back.

short: The ***short keyword*** is used to declare an *integer short variables*.

signed: It is used to declare variables using *signed integers*.

sizeof: The ***sizeof keyword*** is used to calculate the *size (in bytes)* of a data type or object.

static: The ***static keyword*** is used to declare *static variables*, which keep their values across function calls.

struct: A composite *data type* with a single name for all its *variables* is defined with the ***struct keyword***.

switch: It is used to construct a ***switch statement***, which compares an expression to its value and chooses a code block accordingly.

typedef: The ***typedef keyword*** is used to give a *data type* an alias (*alternative name*).

union: It is used to define a *union*, which resembles a *struct* but has one shared memory location for all its members.

Unsigned: The ***unsigned*** keyword is used to define *unsigned integer variables*.

Volatile: This ***keyword*** instructs the compiler that a variable's value may change at any point, even if it is not immediately apparent from the logic of the program.

While: This keyword introduces a loop that repeatedly runs a set of statements if a certain condition is met.

Additional C++ Keywords Not Available in C:

Let's discuss some additional C++ keywords one by one.

asm: The *asm keyword* is used to write programs in *assembly language* that can be inserted into C++ programs. It enables the writing of *inline assembly code* by programmers for certain hardware operations.

dynamic_cast: In C++, *dynamic type casting* uses this keyword. In object-oriented programming, it is mostly used for secure *downcasting* (from a base class to a derived class) when polymorphism is present.

namespace: The *namespace keyword* is used to construct a named scope, which aids in organizing and bringing together similar code pieces. It is very helpful in avoiding name disputes.

reinterpret_cast: In C++, this keyword is used to *type casting* at the lowest level. Without altering the actual data, it may be used to change the type of a pointer to a different type.

bool: The *bool keyword* is used to define the *boolean data type* that can only store *true* or *false values*. It is frequently employed in conditional statements and logical procedures.

explicit: In C++, the *explicit keyword* is used to specify that a *constructor* shouldn't be used automatically to convert types. It reduces the likelihood of accidental automated type conversions.

new: The *heap memory* is used to dynamically allocate memory for objects during runtime. It gives back a pointer to the *RAM* that was allotted.

static_cast: In C++, this keyword is utilized for fundamental *type casting*. It may be used to safely convert between kinds of related types.

false: The *boolean value false* is represented by the term *false*. One of C++'s two boolean literals, the other being true.

catch: The *catch keyword* in *exception handling* is used to capture and manage errors that the *try block throws*. It designates a section of code to run in response to a certain exception being thrown.

operator: In C++, *standard operators* can be overloaded to operate with user-defined types by using the *operator keyword*. It permits the development of *unique operator* behavior.

template: By parameterizing *types* and *functions*, templates are defined by the *template keyword*, enabling *generic programming*. *Templates* provide for flexibility and reuse of code.

friend: A *class* or *function* is designated as a friend of another class using the *friend keyword*. Friends have access to their *classmate's private* information and protected information.

private: The access level of class members is specified using the *private keyword*. Only the class itself has access to private members.

class: In C++, a *class* is defined with the *class keyword*. A user-defined data type called a *class* that contains both *data* and *functions*.

this: It is a reference to the current instance of a class that is referred to by the *term*. Within its *member functions*, it is utilized to access members of the class.

inline: The *inline keyword* is used to instruct the compiler to expand a *function inline*. As a result, the performance can be increased by lowering function call overhead.

public: The access level of *class members* is specified with the *public keyword*. Access to public members is available during the whole program.

throw: The *throw keyword* is used to *manually throw* an exception in C++. It is a crucial component of the C++ exception-handling system.

const_cast: This keyword is used to remove a variable's *const-ness*. It mostly serves to change a variable's type qualifiers.

delete: The memory that was previously allocated using the *new keyword* is released with the *delete keyword*. Memory leaks must be avoided at all costs.

Mutable: In C++, the *mutable keyword* is used to denote the ability to change a member of a *const object*. It is frequently employed when member variables within *const member functions* need to be updated.

protected: The *protected keyword* is used to specify the access level of class members. Accessible inside the *class* and *classes* descended from it are protected members.

truth: The word "*true*" stands for the boolean value "*true*". Along with *false*, it is one of the two boolean literals in C++.

try: In exception handling, the *try keyword* is used to surround a block of code that may *throw exceptions*. One or more catch blocks are placed after it to deal with certain exceptions.

typeid: The *typeid keyword* is used to learn more about an *expression's type*. It is frequently employed for runtime type identification.

typename: In templates, you may use the *typename keyword* to indicate that a dependent name is a *type*. When dealing with template metaprogramming, it is frequently employed.

using: *Namespace aliasing*, *template specialization*, and *declarations* all make use of the keyword. It facilitates *namespace management* and increases code readability.

virtual: The terms *inheritance* and *polymorphism* are used together to describe the concept of virtual. It designates a member function as virtual so that descendant classes may override it.

Wchar_t: A *wide character data type* that can store expanded character sets and support internationalization is defined by the *wchar_t keyword*.

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:

1. `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	== !=	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

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 is 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:

1. Result
2. Test2
3. _sum
4. power

Invalid Identifiers

The following are the examples of invalid identifiers:

1. Sum-1 // containing special character '-'.
2. 2data // the first letter is a digit.
3. **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 not 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.

1. `#include <iostream>`
2. `using namespace std;`
3. `int main()`
4. `{`
5. `int a;`
6. `int A;`
7. `cout<<"Enter the values of 'a' and 'A'";`
8. `cin>>a;`
9. `cin>>A;`
10. `cout<<"\nThe values that you have entered are : "<<a<<" , "<<A;`

11. **return** 0;
12. }

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.

C++ Expression

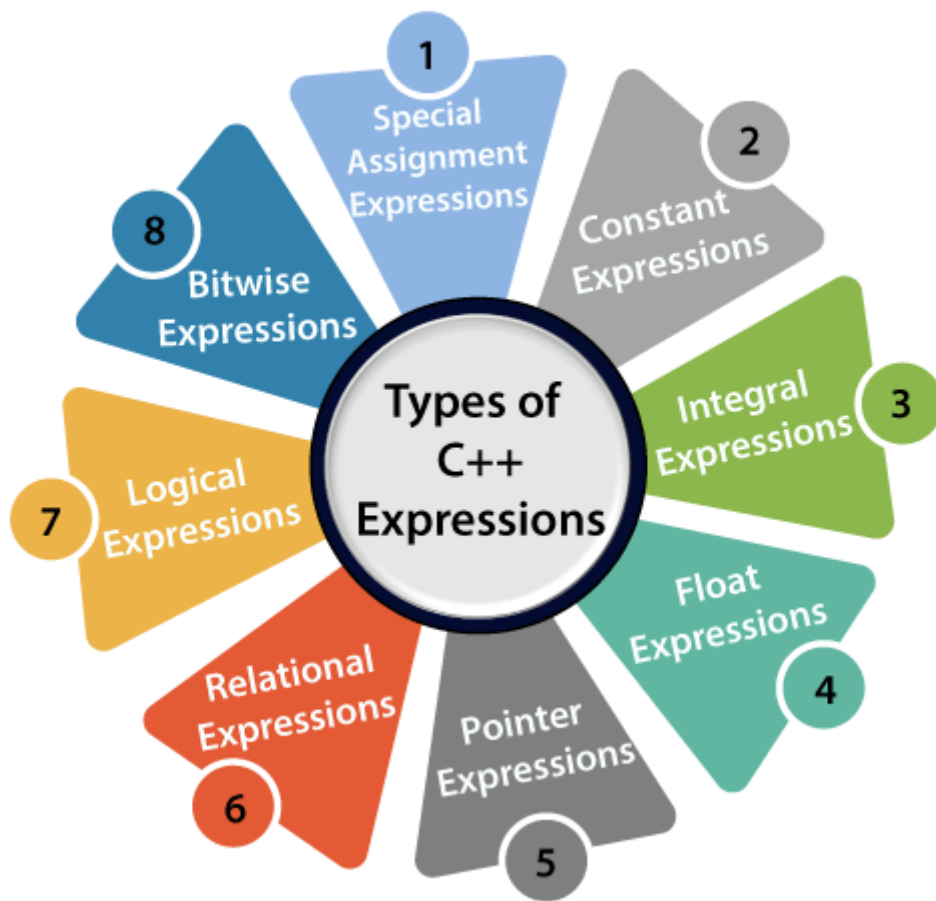
C++ expression consists of operators, constants, and variables which are arranged according to the rules of the language. It can also contain function calls which return values. An expression can consist of one or more operands, zero or more operators to compute a value. Every expression produces some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

Examples of C++ expression:

1. $(a+b) - c$
2. $(x/y) - z$
3. $4a^2 - 5b + c$
4. $(a+b) * (x+y)$

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
--------------------------------	----------------

<code>x = (2/3) * 4</code>	<code>(2/3) * 4</code>
<code>extern int y = 67</code>	<code>67</code>
<code>int z = 43</code>	<code>43</code>
<code>static int a = 56</code>	<code>56</code>

Let's see a simple program containing constant expression:

```

1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int x;    // variable declaration.
6.     x=(3/2) + 2; // constant expression
7.     cout<<"Value of x is : "<<x; // displaying the value of x.
8.     return 0;
9. }
```

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:

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

Let's see a simple example of integral expression:

```

1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int x; // variable declaration.
```

```
6.  int y; // variable declaration
7.  int z; // variable declaration
8.  cout<<"Enter the values of x and y";
9.  cin>>x>>y;
10. z=x+y;
11. cout<<"\n"<<"Value of z is : "<<z; // displaying the value of z.
12. return 0;
13. }
```

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.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.
6.  int x; // variable declaration
7.  int y=9; // variable initialization
8.  x=y+int(10.0); // integral expression
9.  cout<<"Value of x : "<<x; // displaying the value of x.
10. return 0;
11. }
```

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:

ADVERTISEMENT

1. $x+y$
2. $(x/10) + y$
3. 34.5
4. $x+\text{float}(10)$

Let's understand through an example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.
6.     float x=8.9;    // variable initialization
7.     float y=5.6;    // variable initialization
8.     float z;        // variable declaration
9.     z=x+y;
10.    std::cout <<"value of z is :" << z<<std::endl; // displaying the value of z.
11.
12.
13.    return 0;
14. }
```

Output

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

ADVERTISEMENT

ADVERTISEMENT

Let's see another example of float expression.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     float x=6.7;    // variable initialization
6.     float y;        // variable declaration
7.     y=x+float(10);  // float expression
8.     std::cout <<"value of y is :" << y<<std::endl; // displaying the value of y
9.     return 0;
10. }
```

In the above code, we have declared two variables, i.e., x and y. After declaration, we store the value of expression $(x+\text{float}(10))$ 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:

1. &x
2. ptr
3. ptr++
4. ptr-

Let's understand through an example.

1. `#include <iostream>`
2. `using namespace std;`
3. `int main()`
4. `{`
- 5.
6. `int a[]={1,2,3,4,5}; // array initialization`
7. `int *ptr; // pointer declaration`
8. `ptr=a; // assigning base address of array to the pointer ptr`
9. `ptr=ptr+1; // incrementing the value of pointer`
10. `std::cout <<"value of second element of an array : " << *ptr<<std::endl;`
11. `return 0;`
12. `}`

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:

1. a>b
2. a-b >= x-y
3. a+b>80

Let's understand through an example

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int a=45; // variable declaration
6.     int b=78; // variable declaration
7.     bool y= a>b; // relational expression
8.     cout<<"Value of y is :"<<y; // displaying the value of y.
9.     return 0;
10. }
```

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.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int a=4; // variable declaration
6.     int b=5; // variable declaration
7.     int x=3; // variable declaration
8.     int y=6; // variable declaration
9.     cout<<((a+b)>=(x+y)); // relational expression
10. return 0;
11. }
```

In the above code, we have declared four variables, i.e., 'a', 'b', 'x' and 'y'. Then, we apply the relational operator (\geq) 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:

1. `a>b && x>y`
2. `a>10 || b==5`

Let's see a simple example of logical expression.

1. `#include <iostream>`
2. `using namespace std;`
3. `int main()`
4. `{`
5. `int a=2;`
6. `int b=7;`
7. `int c=4;`
8. `cout<<((a>b)||((a>c)));`
9. `return 0;`
10. `}`

Output

0

Bitwise Expressions

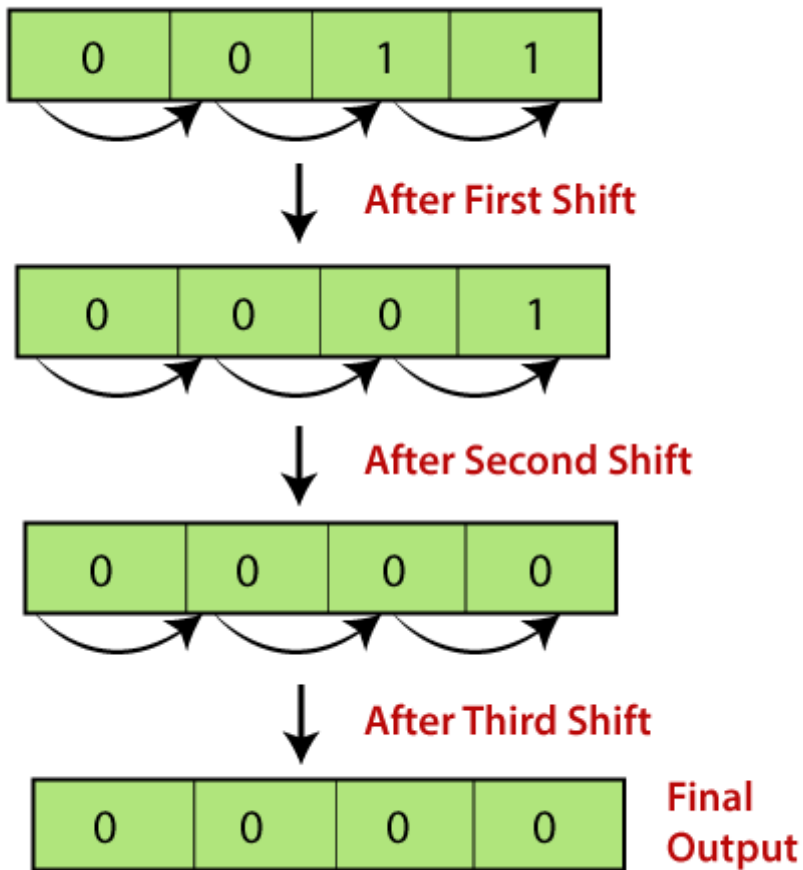
A bitwise expression is an expression which is used to manipulate the data at a bit level. They are basically used to shift the bits.

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.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int x=5; // variable declaration
6.     std::cout << (x>>1) << std::endl;
7.     return 0;
8. }
```

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.

```
1. #include <iostream>
2. using namespace std;
```

```
3. int main()
4. {
5.     int x=7; // variable declaration
6.     std::cout << (x<<3) << std::endl;
7.     return 0;
8. }
```

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:

```
1. a=b=20
2. or
3. (a=b) = 20
```

Let's understand through an example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4.
5.     int a; // variable declaration
6.     int b; // variable declaration
7.     a=b=80; // chained assignment
8.     std::cout << "Values of 'a' and 'b' are : " << a << " " << b << std::endl;
9.     return 0;
10. }
```

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.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int a; // variable declaration
6.     int b; // variable declaration
7.     a=10+(b=90); // embedded assignment expression
8.     std::cout <<"Values of 'a' is " <<a<< std::endl;
9.     return 0;
10. }
```

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,

```
1. a+=10;
```

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

Let's understand through an example.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     int a=10; // variable declaration
6.     a+=10; // compound assignment
```

7. `std::cout << "Value of a is :" <<a<< std::endl; // displaying the value of a.`
8. `return 0;`
9. `}`

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
```