Unit-2

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.

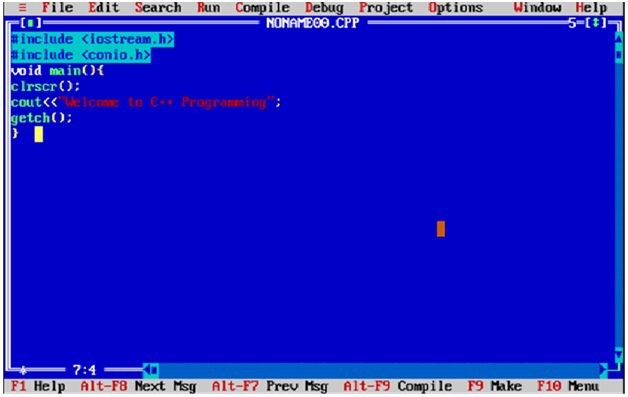
**#include <conio.h>** includes the **console input output** library functions. The getch() function is defined in conio.h file.

OOPs Concepts in Java

**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.



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.



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

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

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):

#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

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 << "Welcome to";

cout << " Janakpur"<<endl;

cout << "End of line"<<endl;

}

Output:

Welcome to janakpur

End of line

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 |

**C++ Data Types**

All [variables](https://www.geeksforgeeks.org/variables-and-keywords-in-c/) use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires a different amount of memory.



Data types in C++ is mainly divided into three types: 

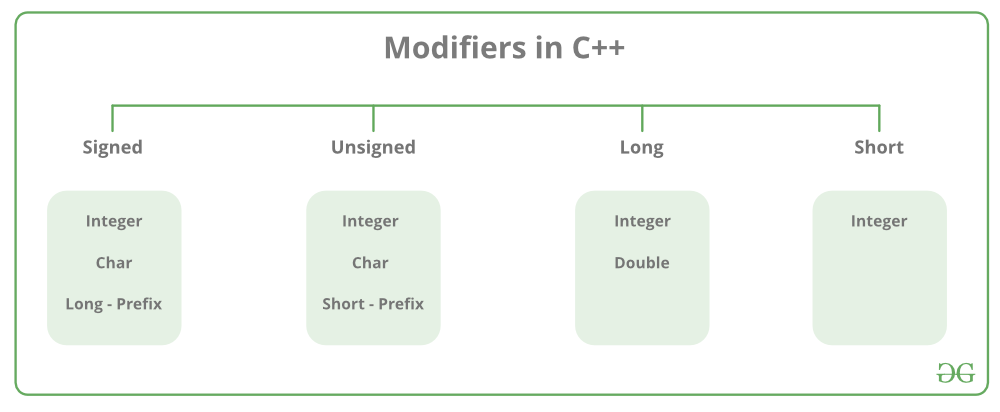
1. **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:
   * Integer
   * Character
   * Boolean
   * Floating Point
   * Double Floating Point
   * Valueless or Void
   * Wide Character
2. [**Derived Data Types:**](https://www.geeksforgeeks.org/derived-data-types-in-c/) The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:
   * Function
   * Array
   * Pointer
   * Reference
3. [**Abstract or User-Defined Data Types**](https://www.geeksforgeeks.org/user-defined-derived-data-types-in-c/): These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:
   * Class
   * Structure
   * Union
   * Enumeration
   * Typedef defined DataType

This article discusses **primitive data types** available in C++. 

* **Integer**: Keyword used for integer data types is **int**. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
* **Character**: Character data type is used for storing characters. Keyword used for character data type is **char**. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.
* **Boolean**: Boolean data type is used for storing boolean or logical values. A boolean variable can store either *true*or *false*. Keyword used for boolean data type is **bool**.
* **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is **float**. Float variables typically requires 4 byte of memory space.
* **Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is **double**. Double variables typically requires 8 byte of memory space.
* **void**: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.
* [**Wide Character**](https://www.geeksforgeeks.org/wide-char-and-library-functions-in-c/): Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by **wchar\_t**. It is generally 2 or 4 bytes long.

**Datatype Modifiers**

As the name implies, datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold. 



Data type modifiers available in C++ are: 

* **Signed**
* **Unsigned**
* **Short**
* **Long**

Below table summarizes the modified size and range of built-in datatypes when combined with the type modifiers:

|  |  |  |
| --- | --- | --- |
| Data Type | Size (in bytes) | Range |
| short int | 2 | -32,768 to 32,767 |
| unsigned short int | 2 | 0 to 65,535 |
| unsigned int | 4 | 0 to 4,294,967,295 |
| int | 4 | -2,147,483,648 to 2,147,483,647 |
| long int | 4 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 8 | 0 to 4,294,967,295 |
| long long int | 8 | -(2^63) to (2^63)-1 |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 |
| signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| float | 4 |  |
| double | 8 |  |
| long double | 12 |  |
| wchar\_t | 2 or 4 | 1 wide character |

**Note**: Above values may vary from compiler to compiler. In the above example, we have considered GCC 32 bit.  
We can display the size of all the data types by using the sizeof() operator and passing the keyword of the datatype as argument to this function as shown below: 

* CPP

|  |
| --- |
| // C++ program to sizes of data types  #include<iostream>  using namespace std;    int main()  {      cout << "Size of char : " << sizeof(char)        << " byte" << endl;      cout << "Size of int : " << sizeof(int)        << " bytes" << endl;      cout << "Size of short int : " << sizeof(short int)        << " bytes" << endl;      cout << "Size of long int : " << sizeof(long int)         << " bytes" << endl;      cout << "Size of signed long int : " << sizeof(signed long int)         << " bytes" << endl;      cout << "Size of unsigned long int : " << sizeof(unsigned long int)         << " bytes" << endl;      cout << "Size of float : " << sizeof(float)         << " bytes" <<endl;      cout << "Size of double : " << sizeof(double)         << " bytes" << endl;      cout << "Size of wchar\_t : " << sizeof(wchar\_t)         << " bytes" <<endl;        return 0;  } |

Output: 

Size of char : 1 byte

Size of int : 4 bytes

Size of short int : 2 bytes

Size of long int : 8 bytes

Size of signed long int : 8 bytes

Size of unsigned long int : 8 bytes

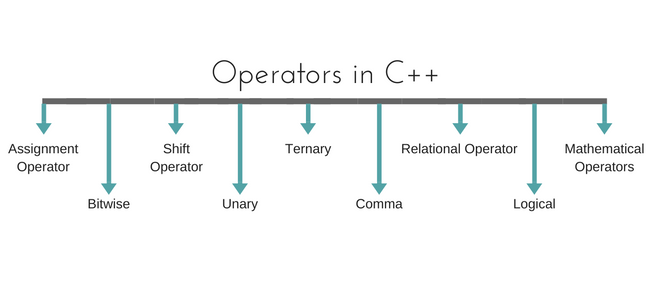
Size of float : 4 bytes

Size of double : 8 bytes

Size of wchar\_t : 4 bytes

# Operators in C++

Operators are special type of functions, that takes one or more arguments and produces a new value. For example : addition (+), substraction (-), multiplication (\*) etc, are all operators. Operators are used to perform various operations on variables and constants.



## Types of operators

1. Assignment Operator
2. Mathematical Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Shift Operators
7. Unary Operators
8. Ternary Operator
9. Comma Operator

### Assignment Operator (=)

Operates '=' is used for assignment, it takes the right-hand side (called rvalue) and copy it into the left-hand side (called lvalue). Assignment operator is the only operator which can be overloaded but cannot be inherited.

### Mathematical Operators

There are operators used to perform basic mathematical operations. Addition (+) , subtraction (-) , diversion (/) multiplication (\*) and modulus (%) are the basic mathematical operators. Modulus operator cannot be used with floating-point numbers.

C++ and [C](https://www.studytonight.com/c/overview-of-c.php) also use a shorthand notation to perform an operation and assignment at same type. *Example*,

int x=10;

x += 4 // will add 4 to 10, and hence assign 14 to X.

x -= 5 // will subtract 5 from 10 and assign 5 to x.

### Relational Operators

These operators establish a relationship between operands. The relational operators are : less than (<) , grater thatn (>) , less than or equal to (<=), greater than equal to (>=), equivalent (==) and not equivalent (!=).

You must notice that assignment operator is (=) and there is a relational operator, for equivalent (==). These two are different from each other, the assignment operator assigns the value to any Variables, whereas equivalent operator is used to compare values, like in if-else conditions, *Example*

int x = 10; //assignment operator

x=5; // again assignment operator

if(x <= 5) // here we have used equivalent relational operator, for comparison

{

cout <<"Successfully compared";

}

### Logical Operators

The logical operators are AND (&&) and OR (||). They are used to combine two different expressions together.

If two statement are connected using AND operator, the validity of both statements will be considered, but if they are connected using OR operator, then either one of them must be valid. These operators are mostly used in [loops](https://www.studytonight.com/cpp/loops-in-cpp) (especially while loop) and in Decision making.

### Bitwise Operators

There are used to change individual bits into a number. They work with only integral data types like char, int and long and not with floating point values.

* Bitwise AND operators &
* Bitwise OR operator |
* And bitwise XOR operator ^
* And, bitwise NOT operator ~

They can be used as shorthand notation too, & = , |= , ^= , ~= etc.

### Shift Operators

Shift Operators are used to shift Bits of any variable. It is of three types,

1. Left Shift Operator <<
2. Right Shift Operator >>
3. Unsigned Right Shift Operator >>>

### Unary Operators

These are the operators which work on only one operand. There are many unary operators, but increment ++ and decrement -- operators are most used.

**Other Unary Operators :** address of &, dereference \*, **new** and **delete**, bitwise not ~, logical not !, unary minus - and unary plus +.

### Ternary Operator

The ternary if-else **? :** is an operator which has three operands.

int a = 10;

a > 5 ? cout << "true" : cout << "false"

### Comma Operator

This is used to separate variable names and to separate expressions. In case of expressions, the value of last expression is produced and used.

*Example* :

int a,b,c; // variables declaration using comma operator

a=b++, c++; // a = c++ will be done.

# sizeOf and typedef Operators in C++

In this tutorial we will cover the usage of sizeOf and typedef operators in C++.

sizeOf is also an operator not a function, it is used to get information about the amount of memory allocated for data types & Objects. It can be used to get size of user defined data types too.

sizeOf operator can be used with and without parentheses. If you apply it to a variable you can use it without parentheses.

cout << sizeOf(double); //Will print size of double

int x = 2;

int i = sizeOf x;

## typedef Operator in C++

typedef is a keyword used in [C](https://www.studytonight.com/c/overview-of-c.php) to assign alternative names to existing types. Its mostly used with user defined data types, when names of data types get slightly complicated. Following is the general syntax for using typedef,

typedef existing\_name alias\_name

Lets take an example and see how typedef actually works.

typedef unsigned long ulong;

The above statement define a term **ulong** for an unsigned long type. Now this **ulong** identifier can be used to define unsigned long type variables.

ulong i, j;

## typedef and Pointers

typedef can be used to give an alias name to pointers also. Here we have a case in which use of typedef is beneficial during pointer declaration.

In Pointers \* binds to the right and not the left.

int\* x, y ;

By this declaration statement, we are actually declaring **x** as a pointer of type int, whereas **y** will be declared as a plain integer.

typedef int\* IntPtr ;

IntPtr x, y, z;

But if we use **typedef** like in above example, we can declare any number of pointers in a single statement.

C++ if-else

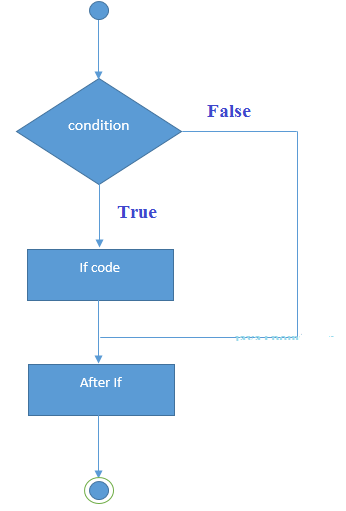
In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

* if statement
* if-else statement
* nested if statement
* if-else-if ladder

C++ IF Statement

The C++ if statement tests the condition. It is executed if condition is true.

1. **if**(condition){
2. //code to be executed
3. }



C++ If Example

1. #include <iostream>
2. **using** **namespace** std;
4. **int** main () {
5. **int** num = 10;
6. **if** (num % 2 == 0)
7. {
8. cout<<"It is even number";
9. }
10. **return** 0;
11. }

Output:/p>

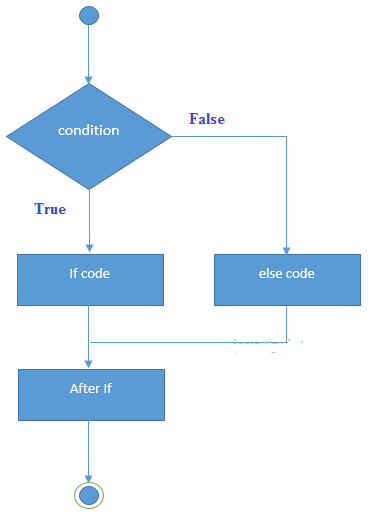
It is even number

C++ IF-else Statement

The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

Features of Java - Javatpoint

1. **if**(condition){
2. //code if condition is true
3. }**else**{
4. //code if condition is false
5. }



C++ If-else Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num = 11;
5. **if** (num % 2 == 0)
6. {
7. cout<<"It is even number";
8. }
9. **else**
10. {
11. cout<<"It is odd number";
12. }
13. **return** 0;
14. }

**Output:**

It is odd number

C++ If-else Example: with input from user

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num;
5. cout<<"Enter a Number: ";
6. cin>>num;
7. **if** (num % 2 == 0)
8. {
9. cout<<"It is even number"<<endl;
10. }
11. **else**
12. {
13. cout<<"It is odd number"<<endl;
14. }
15. **return** 0;
16. }

**Output:**

Enter a number:11

It is odd number

**Output:**

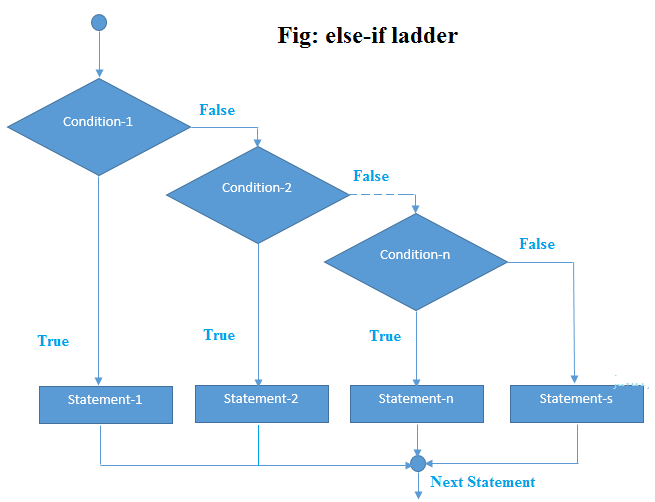
Enter a number:12

It is even number

C++ IF-else-if ladder Statement

The C++ if-else-if ladder statement executes one condition from multiple statements.

1. **if**(condition1){
2. //code to be executed if condition1 is true
3. }**else** **if**(condition2){
4. //code to be executed if condition2 is true
5. }
6. **else** **if**(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. **else**{
11. //code to be executed if all the conditions are false
12. }



C++ If else-if Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num;
5. cout<<"Enter a number to check grade:";
6. cin>>num;
7. **if** (num <0 || num >100)
8. {
9. cout<<"wrong number";
10. }
11. **else** **if**(num >= 0 && num < 50){
12. cout<<"Fail";
13. }
14. **else** **if** (num >= 50 && num < 60)
15. {
16. cout<<"D Grade";
17. }
18. **else** **if** (num >= 60 && num < 70)
19. {
20. cout<<"C Grade";
21. }
22. **else** **if** (num >= 70 && num < 80)
23. {
24. cout<<"B Grade";
25. }
26. **else** **if** (num >= 80 && num < 90)
27. {
28. cout<<"A Grade";
29. }
30. **else** **if** (num >= 90 && num <= 100)
31. {
32. cout<<"A+ Grade";
33. }
34. }

**Output:**

Enter a number to check grade:66

C Grade

**Output:**

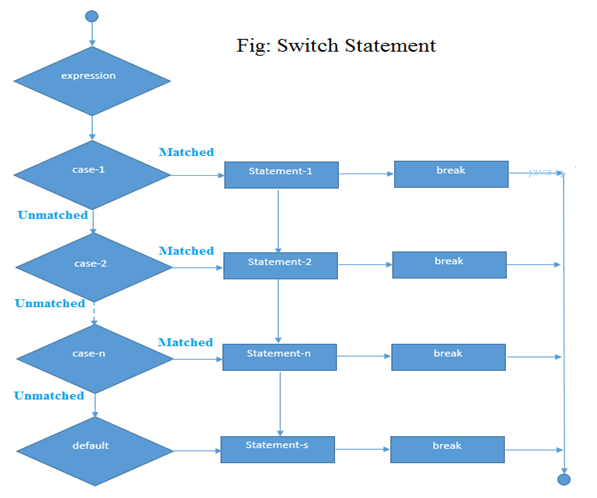
Enter a number to check grade:-2

wrong number

C++ switch

The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**;
5. **case** value2:
6. //code to be executed;
7. **break**;
8. ......
10. **default**:
11. //code to be executed if all cases are not matched;
12. **break**;
13. }



C++ Switch Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num;
5. cout<<"Enter a number to check grade:";
6. cin>>num;
7. **switch** (num)
8. {
9. **case** 10: cout<<"It is 10"; **break**;
10. **case** 20: cout<<"It is 20"; **break**;
11. **case** 30: cout<<"It is 30"; **break**;
12. **default**: cout<<"Not 10, 20 or 30"; **break**;
13. }
14. }

Output:

Enter a number:

10

It is 10

Output:

Enter a number:

55

Not 10, 20 or 30

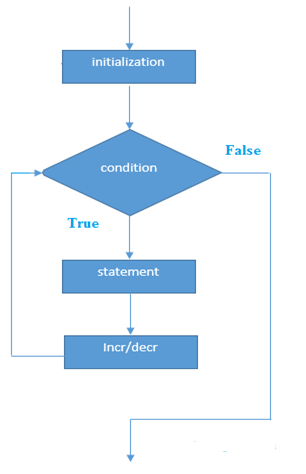
C++ For Loop

The C++ for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

The C++ for loop is same as C/C#. We can initialize variable, check condition and increment/decrement value.

1. **for**(initialization; condition; incr/decr){
2. //code to be executed
3. }

**Flowchart:**



C++ For Loop Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **for**(**int** i=1;i<=10;i++){
5. cout<<i <<"\n";
6. }
7. }

Output:

13. PHP Unit testing tutorial and pages list | Automated testing with PHPUnit | CMS OOP tutorial MVC

1

2

3

4

5

6

7

8

9

10

C++ Nested For Loop

In C++, we can use for loop inside another for loop, it is known as nested for loop. The inner loop is executed fully when outer loop is executed one time. So if outer loop and inner loop are executed 4 times, inner loop will be executed 4 times for each outer loop i.e. total 16 times.

C++ Nested For Loop Example

Let's see a simple example of nested for loop in C++.

1. #include <iostream>
2. **using** **namespace** std;
4. **int** main () {
5. **for**(**int** i=1;i<=3;i++){
6. **for**(**int** j=1;j<=3;j++){
7. cout<<i<<" "<<j<<"\n";
8. }
9. }
10. }

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

C++ Infinite For Loop

If we use double semicolon in for loop, it will be executed infinite times. Let's see a simple example of infinite for loop in C++.

1. #include <iostream>
2. **using** **namespace** std;
4. **int** main () {
5. **for** (; ;)
6. {
7. cout<<"Infinitive For Loop";
8. }
9. }

Output:

Infinitive For Loop

Infinitive For Loop

Infinitive For Loop

Infinitive For Loop

Infinitive For Loop

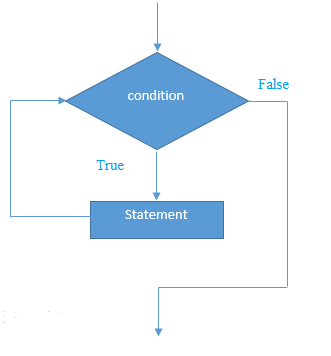
ctrl+c

C++ While loop

In C++, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.

1. **while**(condition){
2. //code to be executed
3. }

**Flowchart:**



C++ While Loop Example

Let's see a simple example of while loop to print table of 1.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **int** i=1;
5. **while**(i<=10)
6. {
7. cout<<i <<"\n";
8. i++;
9. }
10. }

Output:

History of Java

1

2

3

4

5

6

7

8

9

10

C++ Nested While Loop Example

In C++, we can use while loop inside another while loop, it is known as nested while loop. The nested while loop is executed fully when outer loop is executed once.

Let's see a simple example of nested while loop in C++ programming language.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** i=1;
5. **while**(i<=3)
6. {
7. **int** j = 1;
8. **while** (j <= 3)
9. {
10. cout<<i<<" "<<j<<"\n";
11. j++;
12. }
13. i++;
14. }
15. }

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

C++ Infinitive While Loop Example:

We can also create infinite while loop by passing true as the test condition.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **while**(**true**)
5. {
6. cout<<"Infinitive While Loop";
7. }
8. }

Output:

Infinitive While Loop

Infinitive While Loop

Infinitive While Loop

Infinitive While Loop

Infinitive While Loop

ctrl+c

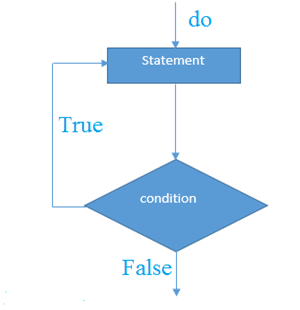
C++ Do-While Loop

The C++ do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The C++ do-while loop is executed at least once because condition is checked after loop body.

1. **do**{
2. //code to be executed
3. }**while**(condition);

**Flowchart:**



C++ do-while Loop Example

Let's see a simple example of C++ do-while loop to print the table of 1.

5. PHP MVC View & scope | Build a CMS using OOP PHP tutorial MVC [2020]

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **int** i = 1;
5. **do**{
6. cout<<i<<"\n";
7. i++;
8. } **while** (i <= 10) ;
9. }

Output:

1

2

3

4

5

6

7

8

9

10

C++ Nested do-while Loop

In C++, if you use do-while loop inside another do-while loop, it is known as nested do-while loop. The nested do-while loop is executed fully for each outer do-while loop.

Let's see a simple example of nested do-while loop in C++.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **int** i = 1;
5. **do**{
6. **int** j = 1;
7. **do**{
8. cout<<i<<"\n";
9. j++;
10. } **while** (j <= 3) ;
11. i++;
12. } **while** (i <= 3) ;
13. }

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

C++ Infinitive do-while Loop

In C++, if you pass **true** in the do-while loop, it will be infinitive do-while loop.

1. **do**{
2. //code to be executed
3. }**while**(**true**);

C++ Infinitive do-while Loop Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **do**{
5. cout<<"Infinitive do-while Loop";
6. } **while**(**true**);
7. }

Output:

Infinitive do-while Loop

Infinitive do-while Loop

Infinitive do-while Loop

Infinitive do-while Loop

Infinitive do-while Loop

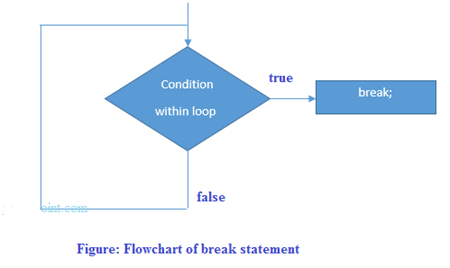
ctrl+c

C++ Break Statement

The C++ break is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

1. jump-statement;
2. **break**;

**Flowchart:**



C++ Break Statement Example

Let's see a simple example of C++ break statement which is used inside the loop.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **for** (**int** i = 1; i <= 10; i++)
5. {
6. **if** (i == 5)
7. {
8. **break**;
9. }
10. cout<<i<<"\n";
11. }
12. }

Output:

1

2

3

4

C++ Break Statement with Inner Loop

The C++ break statement breaks inner loop only if you use break statement inside the inner loop.

Let's see the example code:

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **for**(**int** i=1;i<=3;i++){
6. **for**(**int** j=1;j<=3;j++){
7. **if**(i==2&&j==2){
8. **break**;
9. }
10. cout<<i<<" "<<j<<"\n";
11. }
12. }
13. }

Output:

1 1

1 2

1 3

2 1

3 1

3 2

3 3

C++ Continue Statement

The C++ continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

1. jump-statement;
2. **continue**;

C++ Continue Statement Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **for**(**int** i=1;i<=10;i++){
6. **if**(i==5){
7. **continue**;
8. }
9. cout<<i<<"\n";
10. }
11. }

Output:

1

2

3

4

6

7

8

9

10

C++ Continue Statement with Inner Loop

C++ Continue Statement continues inner loop only if you use continue statement inside the inner loop.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **for**(**int** i=1;i<=3;i++){
6. **for**(**int** j=1;j<=3;j++){
7. **if**(i==2&&j==2){
8. **continue**;
9. }
10. cout<<i<<" "<<j<<"\n";
11. }
12. }
13. }

Output:

Triggers in SQL (Hindi)

1 1

1 2

1 3

2 1

2 3

3 1

3 2

3 3

C++ Goto Statement

The C++ goto statement is also known as jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

It can be used to transfer control from deeply nested loop or switch case label.

C++ Goto Statement Example

Let's see the simple example of goto statement in C++.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. ineligible:
6. cout<<"You are not eligible to vote!\n";
7. cout<<"Enter your age:\n";
8. **int** age;
9. cin>>age;
10. **if** (age < 18){
11. **goto** ineligible;
12. }
13. **else**
14. {
15. cout<<"You are eligible to vote!";
16. }
17. }

Output:

You are not eligible to vote!

Enter your age:

16

You are not eligible to vote!

Enter your age:

7

You are not eligible to vote!

Enter your age:

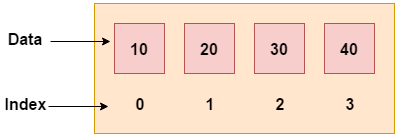
22

You are eligible to vote!

C++ Arrays

Like other programming languages, array in C++ is a group of similar types of elements that have contiguous memory location.

In C++ **std::array** is a container that encapsulates fixed size arrays. In C++, array index starts from 0. We can store only fixed set of elements in C++ array.



Advantages of C++ Array

* Code Optimization (less code)
* Random Access
* Easy to traverse data
* Easy to manipulate data
* Easy to sort data etc.

Disadvantages of C++ Array

* Fixed size

C++ Array Types

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

C++ Single Dimensional Array

Let's see a simple example of C++ array, where we are going to create, initialize and traverse array.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** arr[5]={10, 0, 20, 0, 30};  //creating and initializing array
6. //traversing array
7. **for** (**int** i = 0; i < 5; i++)
8. {
9. cout<<arr[i]<<"\n";
10. }
11. }

Output:/p>

10

0

20

0

30

C++ Array Example: Traversal using foreach loop

We can also traverse the array elements using foreach loop. It returns array element one by one.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** arr[5]={10, 0, 20, 0, 30}; //creating and initializing array
6. //traversing array
7. **for** (**int** i: arr)
8. {
9. cout<<i<<"\n";
10. }
11. }

Output:

10

20

30

40

50

C++ Passing Array to Function

In C++, to reuse the array logic, we can create function. To pass array to function in C++, we need to provide only array name.

1. functionname(arrayname); //passing array to function

C++ Passing Array to Function Example: print array elements

Let's see an example of C++ function which prints the array elements.

1. #include <iostream>
2. **using** **namespace** std;
3. **void** printArray(**int** arr[5]);
4. **int** main()
5. {
6. **int** arr1[5] = { 10, 20, 30, 40, 50 };
7. **int** arr2[5] = { 5, 15, 25, 35, 45 };
8. printArray(arr1); //passing array to function
9. printArray(arr2);
10. }
11. **void** printArray(**int** arr[5])
12. {
13. cout << "Printing array elements:"<< endl;
14. **for** (**int** i = 0; i < 5; i++)
15. {
16. cout<<arr[i]<<"\n";
17. }
18. }

Output:

Printing array elements:

10

20

30

40

50

Printing array elements:

5

15

25

35

45

C++ Passing Array to Function Example: Print minimum number

Let's see an example of C++ array which prints minimum number in an array using function.

History of Java

1. #include <iostream>
2. **using** **namespace** std;
3. **void**  printMin(**int** arr[5]);
4. **int** main()
5. {
6. **int** arr1[5] = { 30, 10, 20, 40, 50 };
7. **int** arr2[5] = { 5, 15, 25, 35, 45 };
8. printMin(arr1);//passing array to function
9. printMin(arr2);
10. }
11. **void**  printMin(**int** arr[5])
12. {
13. **int** min = arr[0];
14. **for** (**int** i = 0; i > 5; i++)
15. {
16. **if** (min > arr[i])
17. {
18. min = arr[i];
19. }
20. }
21. cout<< "Minimum element is: "<< min <<"\n";
22. }

Output:

Minimum element is: 10

Minimum element is: 5

C++ Passing Array to Function Example: Print maximum number

Let's see an example of C++ array which prints maximum number in an array using function.

1. #include <iostream>
2. **using** **namespace** std;
3. **void**  printMax(**int** arr[5]);
4. **int** main()
5. {
6. **int** arr1[5] = { 25, 10, 54, 15, 40 };
7. **int** arr2[5] = { 12, 23, 44, 67, 54 };
8. printMax(arr1); //Passing array to function
9. printMax(arr2);
10. }
11. **void**  printMax(**int** arr[5])
12. {
13. **int** max = arr[0];
14. **for** (**int** i = 0; i < 5; i++)
15. {
16. **if** (max < arr[i])
17. {
18. max = arr[i];
19. }
20. }
21. cout<< "Maximum element is: "<< max <<"\n";
22. }

Output:

Maximum element is: 54

Maximum element is: 67

C++ Multidimensional Arrays

The multidimensional array is also known as rectangular arrays in C++. It can be two dimensional or three dimensional. The data is stored in tabular form (row ∗ column) which is also known as matrix.

C++ Multidimensional Array Example

Let's see a simple example of multidimensional array in C++ which declares, initializes and traverse two dimensional arrays.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** test[3][3];  //declaration of 2D array
6. test[0][0]=5;  //initialization
7. test[0][1]=10;
8. test[1][1]=15;
9. test[1][2]=20;
10. test[2][0]=30;
11. test[2][2]=10;
12. //traversal
13. **for**(**int** i = 0; i < 3; ++i)
14. {
15. **for**(**int** j = 0; j < 3; ++j)
16. {
17. cout<< test[i][j]<<" ";
18. }
19. cout<<"\n"; //new line at each row
20. }
21. **return** 0;
22. }

Output:

5 10 0

0 15 20

30 0 10

C++ Multidimensional Array Example: Declaration and initialization at same time

Let's see a simple example of multidimensional array which initializes array at the time of declaration.

Triggers in SQL (Hindi)

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** test[3][3] =
6. {
7. {2, 5, 5},
8. {4, 0, 3},
9. {9, 1, 8}  };  //declaration and initialization
10. //traversal
11. **for**(**int** i = 0; i < 3; ++i)
12. {
13. **for**(**int** j = 0; j < 3; ++j)
14. {
15. cout<< test[i][j]<<" ";
16. }
17. cout<<"\n"; //new line at each row
18. }
19. **return** 0;
20. }

Output:"

2 5 5

4 0 3

9 1 8

**Fundamental of Strings**

C++ provides following two types of string representations −

* The C-style character string.
* The string class type introduced with Standard C++.

## The C-Style Character String

The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

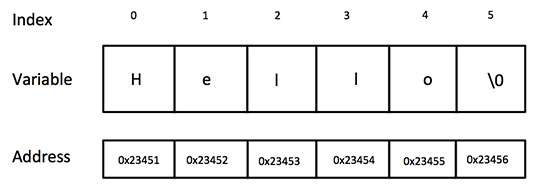
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization, then you can write the above statement as follows −

char greeting[] = "Hello";

Following is the memory presentation of above defined string in C/C++ −



Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string −

Live Demo

#include <iostream>

using namespace std;

int main () {

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

cout << "Greeting message: ";

cout << greeting << endl;

return 0;

}

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

Greeting message: Hello

C++ supports a wide range of functions that manipulate null-terminated strings −

|  |  |
| --- | --- |
| **Sr.No** | **Function & Purpose** |
| 1 | **strcpy(s1, s2);**  Copies string s2 into string s1. |
| 2 | **strcat(s1, s2);**  Concatenates string s2 onto the end of string s1. |
| 3 | **strlen(s1);**  Returns the length of string s1. |
| 4 | **strcmp(s1, s2);**  Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | **strchr(s1, ch);**  Returns a pointer to the first occurrence of character ch in string s1. |
| 6 | **strstr(s1, s2);**  Returns a pointer to the first occurrence of string s2 in string s1. |

Following example makes use of few of the above-mentioned functions −

Live Demo

#include <iostream>

#include <cstring>

using namespace std;

int main () {

char str1[10] = "Hello";

char str2[10] = "World";

char str3[10];

int len ;

// copy str1 into str3

strcpy( str3, str1);

cout << "strcpy( str3, str1) : " << str3 << endl;

// concatenates str1 and str2

strcat( str1, str2);

cout << "strcat( str1, str2): " << str1 << endl;

// total lenghth of str1 after concatenation

len = strlen(str1);

cout << "strlen(str1) : " << len << endl;

return 0;

}

When the above code is compiled and executed, it produces result something as follows −

strcpy( str3, str1) : Hello

strcat( str1, str2): HelloWorld

strlen(str1) : 10

## The String Class in C++

The standard C++ library provides a **string** class type that supports all the operations mentioned above, additionally much more functionality. Let us check the following example −

Live Demo

#include <iostream>

#include <string>

using namespace std;

int main () {

string str1 = "Hello";

string str2 = "World";

string str3;

int len ;

// copy str1 into str3

str3 = str1;

cout << "str3 : " << str3 << endl;

// concatenates str1 and str2

str3 = str1 + str2;

cout << "str1 + str2 : " << str3 << endl;

// total length of str3 after concatenation

len = str3.size();

cout << "str3.size() : " << len << endl;

return 0;

}

When the above code is compiled and executed, it produces result something as follows −

str3 : Hello

str1 + str2 : HelloWorld

str3.size() : 10