

History of C++

01 December 2022 16:32

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

A header file contains:

1. Function definitions
2. Data type definitions
3. Macros

It offers the above features by importing them into the program with the help of a preprocessor directive “**#include**”. These preprocessor directives are used for instructing compiler that these files need to be processed before compilation.

Standard Header Files And Their Uses:

4. **#include<stdio.h>**: It is used to perform input and output operations using functions **scanf()** and **printf()**.
5. **#include<iostream>**: It is used as a stream of Input and Output using **cin** and **cout**.
6. **#include<string.h>**: It is used to perform various functionalities related to string manipulation like [strlen\(\)](#), [strcmp\(\)](#), [strcpy\(\)](#), [size\(\)](#), etc.
7. **#include<math.h>**: It is used to perform mathematical operations like [sqrt\(\)](#), [log2\(\)](#), [pow\(\)](#), etc.
8. **#include<iomanip.h>**: It is used to access **set()** and **setprecision()** function to limit the decimal places in variables.
9. **#include<signal.h>**: It is used to perform signal handling functions like **signal()** and **raise()**.
10. **#include<stdarg.h>**: It is used to perform standard argument functions like **va_start()** and **va_arg()**. It is also used to indicate start of the variable-length argument list and to fetch the arguments from the variable-length argument list in the program respectively.
11. **#include<errno.h>**: It is used to perform [error handling](#) operations like **errno()**, **strerror()**, **perror()**, etc.
12. **#include<fstream.h>**: It is used to control the data to read from a file as an input and data to write into the file as an output.
13. **#include<time.h>**: It is used to perform functions related to date() and [time\(\)](#) like [setdate\(\)](#) and [getdate\(\)](#). It is also used to modify the system date and get the CPU time respectively.
14. **#include<float.h>**: It contains a set of various platform-dependent constants related to floating point values. These constants are proposed by ANSI C. They allow making programs more portable. Some examples of constants included in this header file are- **e**(exponent), **b**(base/radix), etc.
15. **#include<limits.h>**: It determines various properties of the various variable types. The macros defined in this header, limits the values of various variable types like **char**, **int**, and **long**. These limits specify that a variable cannot store any value beyond these limits, for example an unsigned character can store up to a maximum value of **255**.
16. **#include<assert.h>**: It contains information for adding diagnostics that aid program debugging.
17. **#include<ctype.h>**: It contains function prototypes for functions that test characters for certain properties , and also function prototypes for functions that can be used to convert uppercase letters to lowercase letters and vice versa.
18. **#include<locale.h>**: It contains function prototypes and other information that enables a program to be modified for the current locale on which it's running. It enables the computer system to handle different conventions for expressing data such as times, dates or large numbers throughout the world.
19. **#include<setjmp.h>**: It contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
20. **#include<stddef.h>**: It contains common type definitions used by C for performing calculations.

Non-Standard Header File And its Uses:

- **#include<bits/stdc++.h>**: It contains *all standard library* of the header files mentioned above. So if you include it in your code, then you need not have to include any other standard header files. But as it is a non-standard header file of GNU C++ library, so, if you try to compile your code with some compiler other than GCC it might fail; e.g. MSVC do not have this header.

Namespace

01 December 2022 16:46

- Namespace provide the space where we can define or declare identifier i.e. variable, method, classes.
- Using namespace, you can define the space or context in which identifiers are defined i.e. variable, method, classes. In essence, a namespace defines a scope.

Advantage of Namespace to avoid name collision.

- Example, you might be writing some code that has a function called xyz() and there is another library available which is also having same function xyz(). Now the compiler has no way of knowing which version of xyz() function you are referring to within your code.
- A namespace is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries.
- The best example of namespace scope is the C++ standard library (std) where all the classes, methods and templates are declared. Hence while writing a C++ program we usually include the directive using namespace std;

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



Data Types in C++

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	

C++ Operators

In this tutorial, we will learn about the different types of operators in C++ with the help of examples. In programming, an operator is a symbol that operates on a value or a variable.

Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while - is an operator used for subtraction.

Operators in C++ can be classified into 6 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Other Operators

1. C++ Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data. For example,

```
a + b;
```

Here, the + operator is used to add two variables a and b. Similarly there are various other arithmetic operators in C++.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Example 1: Arithmetic Operators

```
#include<iostream>
Using namespace std;
int main(){
    int a, b;
    a = 7;
    b = 2;
    // printing the sum of a and b cout<< "a + b = "<< (a + b) << endl;
    // printing the difference of a and b cout<< "a - b = "<< (a - b) << endl;
    // printing the product of a and b cout<< "a * b = "<< (a * b) << endl;
    // printing the division of a by b cout<< "a / b = "<< (a / b) << endl;
```

```
// printing the modulo of a by bcout<< "a % b = "<< (a % b) << endl;
return0;
}
```

Output

```
a + b = 9
a - b = 5
a * b = 14
a / b = 3
a % b = 1
```

Here, the operators +, - and * compute addition, subtraction, and multiplication respectively as we might have expected.

/ Division Operator

Note the operation (a / b) in our program. The / operator is the division operator.

As we can see from the above example, if an integer is divided by another integer, we will get the quotient. However, if either divisor or dividend is a floating-point number, we will get the result in decimals.

```
In C++,
7/2 is 3
7.0 / 2 is 3.5
7 / 2.0 is 3.5
7.0 / 2.0 is 3.5
```

% Modulo Operator

The modulo operator % computes the remainder. When a = 9 is divided by b = 4, the remainder is 1.

Note: The % operator can only be used with integers.

Increment and Decrement Operators

C++ also provides increment and decrement operators: ++ and -- respectively.

- ++ increases the value of the operand by 1
- -- decreases it by 1

For example,

```
intnum = 5;
// increment operator++num; // 6
```

Here, the code ++num; increases the value of num by 1.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators#include<iostream>usingnamespacestd;
intmain(){
    inta = 10, b = 100, result_a, result_b;
    // incrementing a by 1 and storing the result in result_aresult_a = ++a;
    cout<< "result_a = "<< result_a << endl;
    // decrementing b by 1 and storing the result in result_b result_b = --b;
    cout<< "result_b = "<< result_b << endl;
    return0;
}
```

[Run Code](#)

Output

```
result_a = 11
result_b = 99
```

In the above program, we have used the ++ and -- operators as **prefixes (++a and --b)**. However, we can also use these operators as **postfix (a++ and b--)**.

2. C++ Assignment Operators

In C++, assignment operators are used to assign values to variables. For example,

```
// assign 5 to aa = 5;
```

Here, we have assigned a value of 5 to the variable a.

Operator	Example	Equivalent to
----------	---------	---------------

=	a = b;	a = b;
---	--------	--------

+=	a += b;	a = a + b;
----	---------	------------

-=	a -= b;	a = a - b;
----	---------	------------

*=	a *= b;	a = a * b;
----	---------	------------

/=	a /= b;	a = a / b;
----	---------	------------

%=	a %= b;	a = a % b;
----	---------	------------

Example 3: Assignment Operators

```
#include<iostream>using namespace std;
int main(){
    int a, b;
    // 2 is assigned to aa = 2;
    // 7 is assigned to bb = 7;
    cout<< "a = " << a << endl;
    cout<< "b = " << b << endl;
    cout<< "\nAfter a += b;" << endl;
    // assigning the sum of a and b to aa += b; // a = a + b
    cout<< "a = " << a << endl;
    return 0;
}
```

[Run Code](#)

Output

```
a = 2
b = 7
After a += b;
a = 9
```

3. C++ Relational Operators

A relational operator is used to check the relationship between two operands. For example,

```
// checks if a is greater than b a > b;
```

Here, > is a relational operator. It checks if a is greater than b or not.

If the relation is **true**, it returns **1** whereas if the relation is **false**, it returns **0**.

Operator	Meaning	Example
==	Is Equal To	3 == 5 gives us false
!=	Not Equal To	3 != 5 gives us true
>	Greater Than	3 > 5 gives us false
<	Less Than	3 < 5 gives us true
>=	Greater Than or Equal To	3 >= 5 give us false
<=	Less Than or Equal To	3 <= 5 gives us true

Example 4: Relational Operators

```
#include<iostream>usingnamespacestd;
intmain(){
    inta, b;
    a = 3;
    b = 5;
    boolresult;
    result = (a == b); // falsecout<< "3 == 5 is "<< result << endl;
    result = (a != b); // truecout<< "3 != 5 is "<< result << endl;
    result = a > b; // falsecout<< "3 > 5 is "<< result << endl;
    result = a < b; // truecout<< "3 < 5 is "<< result << endl;
    result = a >= b; // falsecout<< "3 >= 5 is "<< result << endl;
    result = a <= b; // truecout<< "3 <= 5 is "<< result << endl;
    return0;
}
```

[Run Code](#)

Output

```
3 == 5 is 0
3 != 5 is 1
3 > 5 is 0
3 < 5 is 1
3 >= 5 is 0
3 <= 5 is 1
```

Note: Relational operators are used in decision-making and loops.

4. C++ Logical Operators

Logical operators are used to check whether an expression is **true** or **false**. If the expression is **true**, it returns **1** whereas if the expression is **false**, it returns **0**.

Operator	Example	Meaning
&&	expression1 && expression2	Logical AND. True only if all the operands are true.
	expression1 expression2	Logical OR. True if at least one of the operands is true.
!	!expression	Logical NOT. True only if the operand is false.

In C++, logical operators are commonly used in decision making. To further understand the logical operators, let's see the following examples,

```
Suppose,
a = 5 b = 8Then,
(a > 3) && (b > 5) evaluates to true(a > 3) && (b < 5) evaluates to false(a > 3) || (b > 5) evaluates to true(a > 3) || (b < 5) evaluates to true(a < 3) || (b < 5) evaluates to false!(a < 3) evaluates to true!(a > 3) evaluates to false
```

Example 5: Logical Operators

```
#include<iostream>usingnamespacestd;
intmain(){
    boolresult;
    result = (3!= 5) && (3< 5); // truecout<< "(3 != 5) && (3 < 5) is "<< result << endl;
    result = (3== 5) && (3< 5); // falsecout<< "(3 == 5) && (3 < 5) is "<< result << endl;
    result = (3== 5) && (3> 5); // falsecout<< "(3 == 5) && (3 > 5) is "<< result << endl;
    result = (3!= 5) || (3< 5); // truecout<< "(3 != 5) || (3 < 5) is "<< result << endl;
    result = (3!= 5) || (3> 5); // truecout<< "(3 != 5) || (3 > 5) is "<< result << endl;
    result = (3== 5) || (3> 5); // falsecout<< "(3 == 5) || (3 > 5) is "<< result << endl;
    result = !(5== 2); // truecout<< "!(5 == 2) is "<< result << endl;
    result = !(5== 5); // falsecout<< "!(5 == 5) is "<< result << endl;
    return0;
}
```

```
}
```

[Run Code](#)

Output

```
(3 != 5) && (3 < 5) is 1
(3 == 5) && (3 < 5) is 0
(3 == 5) && (3 > 5) is 0
(3 != 5) || (3 < 5) is 1
(3 != 5) || (3 > 5) is 1
(3 == 5) || (3 > 5) is 0
!(5 == 2) is 1
!(5 == 5) is 0
```

Explanation of logical operator program

- `(3 != 5) && (3 < 5)` evaluates to **1** because both operands `(3 != 5)` and `(3 < 5)` are **1** (true).
- `(3 == 5) && (3 < 5)` evaluates to **0** because the operand `(3 == 5)` is **0** (false).
- `(3 == 5) && (3 > 5)` evaluates to **0** because both operands `(3 == 5)` and `(3 > 5)` are **0** (false).
- `(3 != 5) || (3 < 5)` evaluates to **1** because both operands `(3 != 5)` and `(3 < 5)` are **1** (true).
- `(3 != 5) || (3 > 5)` evaluates to **1** because the operand `(3 != 5)` is **1** (true).
- `(3 == 5) || (3 > 5)` evaluates to **0** because both operands `(3 == 5)` and `(3 > 5)` are **0** (false).
- `!(5 == 2)` evaluates to **1** because the operand `(5 == 2)` is **0** (false).
- `!(5 == 5)` evaluates to **0** because the operand `(5 == 5)` is **1** (true).

5. C++ Bitwise Operators

In C++, bitwise operators are used to perform operations on individual bits. They can only be used alongside `char` and `int` data types.

Operator	Description
<code>&</code>	Binary AND
<code> </code>	Binary OR
<code>^</code>	Binary XOR
<code>~</code>	Binary One's Complement
<code><<</code>	Binary Shift Left
<code>>></code>	Binary Shift Right

6. Other C++ Operators

Here's a list of some other common operators available in C++. We will learn about them in later tutorials.

Operator	Description	Example
<code>sizeof</code>	returns the size of data type	<code>sizeof(int); // 4</code>
<code>?:</code>	returns value based on the condition	<code>string result = (5 > 0) ? "even" : "odd"; // "even"</code>
<code>&</code>	represents memory address of the operand	<code>&num; // address of num</code>
<code>.</code>	accesses members of struct variables or class objects	<code>s1.marks = 92;</code>
<code>-></code>	used with pointers to access the class or struct variables	<code>ptr->marks = 92;</code>
<code><<</code>	prints the output value	<code>cout << 5;</code>
<code>>></code>	gets the input value	<code>cin >> num;</code>

C++ CONDITIONAL STATEMENTS

01 December 2022 16:56

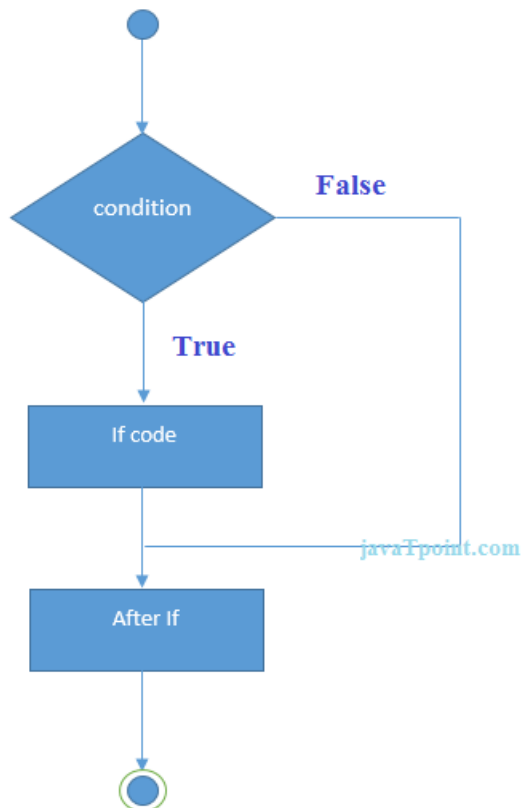
In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

- 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;  
3.  
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

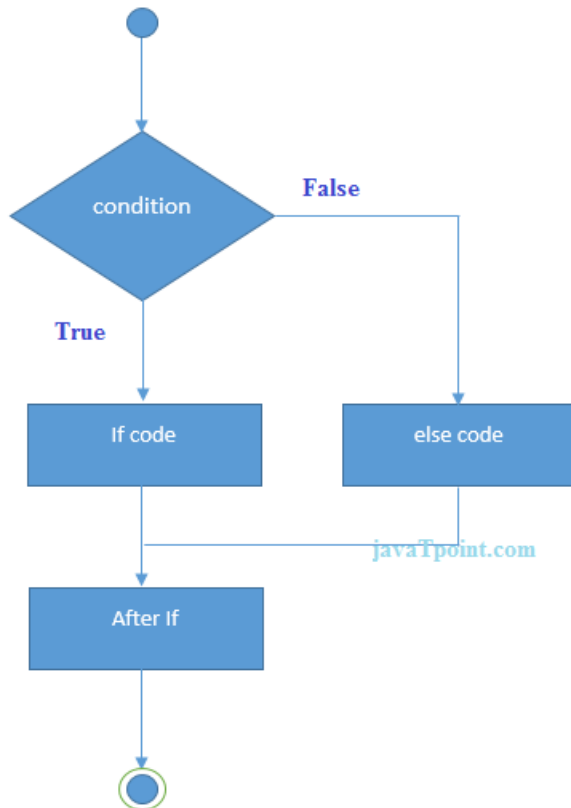
The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

```
1. if(condition){  
2. //code if condition is true
```

```

3. }else{
4. //code if condition is false
5. }

```



AD

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

```

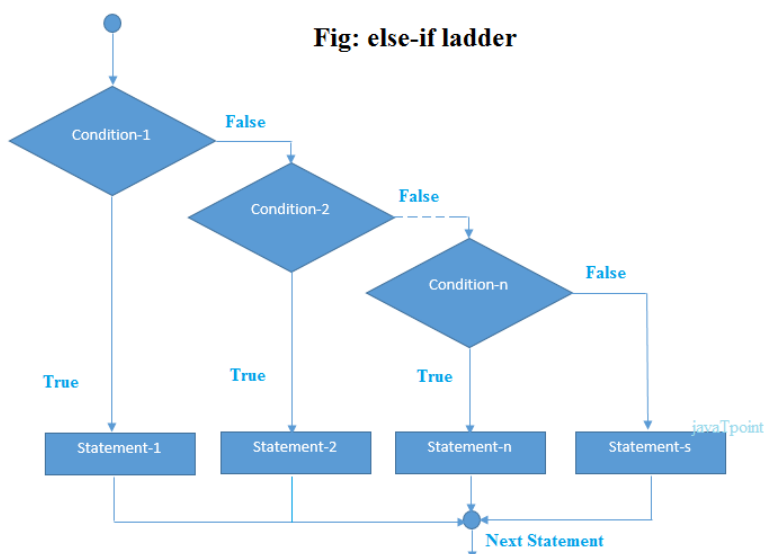
C++ IF-else-if ladder Statement

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

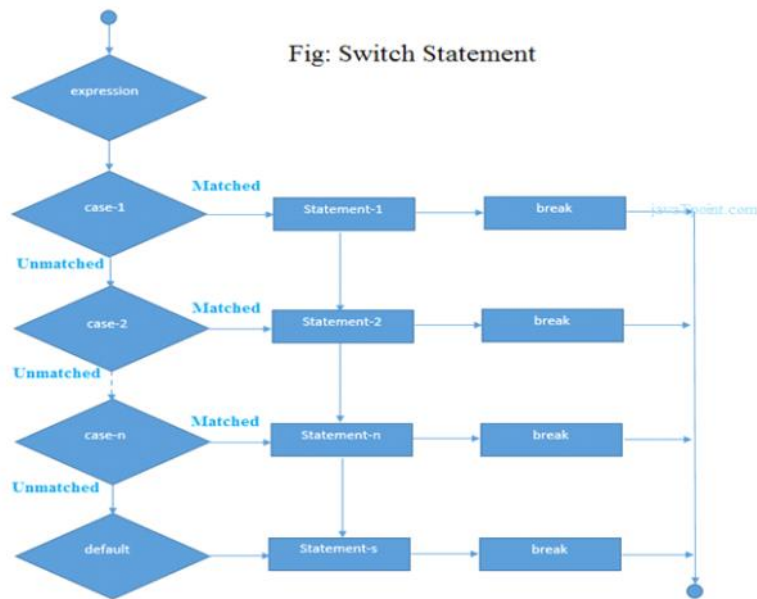
Output:

Enter a number to check grade:-2
wrong number

C++	SWITCH-CASE STATEMENTS
-----	-------------------------------

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.
- 9.
10. **default:**
11. *//code to be executed if all cases are not matched;*
12. **break;**
13. }



C++ Switch Example

```

14. #include <iostream>
15. using namespace std;
16. int main () {
17.     int num;
18.     cout<<"Enter a number to check grade:";
19.     cin>>num;
20.     switch (num)
21.     {
22.         case 10: cout<<"It is 10"; break;
23.         case 20: cout<<"It is 20"; break;
24.         case 30: cout<<"It is 30"; break;
25.         default: cout<<"Not 10, 20 or 30"; break;
26.     }
27. }
  
```

Output:

Enter a number:

10

It is 10

Output:

Enter a number:

55

Not 10, 20 or 30

C++ Conditional Exercises

01 December 2022 17:02

C++ Program to Check Whether Number is Even or Odd

C++ Program to Check Whether a character is Vowel or Consonant.

C++ Program to Find Largest Number Among Three Numbers

Hacker rank: [Solve C++ | HackerRank](#)

In programming, sometimes there is a need to perform some operation **more than once** or (say) **n number** of times. Loops come into use when we need to repeatedly execute a block of statements.

For example: Suppose we want to print “Hello World” 10 times. This can be done in two ways as shown below:

Manual Method (Iterative Method)

Manually we have to write **cout** for the C++ statement 10 times. Let's say you have to write it 20 times (it would surely take more time to write 20 statements) now imagine you have to write it 100 times, it would be really hectic to re-write the same statement again and again. So, *here loops have their role*.

- C++

```
// C++ program to Demonstrate the need of loops
#include <iostream>
using namespace std;

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

Output

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

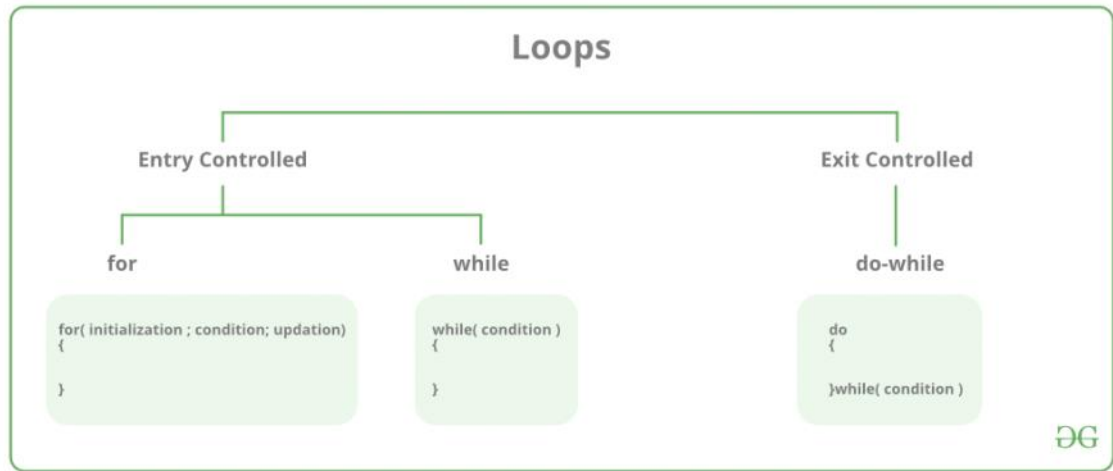
Using Loops

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below. In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

- An operation is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.
- **Counter not Reached:** If the counter has not reached the desired number, the next instruction in the sequence returns to the first instruction in the sequence and repeats it.
- **Counter reached:** If the condition has been reached, the next instruction “falls through” to the next sequential instruction or branches outside the loop.

There are mainly two types of loops:

1. **Entry Controlled loops:** In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loop** is entry-controlled loops.
2. **Exit Controlled Loops:** In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do-while **loop** is exit controlled loop.



S.No. Loop Type and Description

1. **while loop** – First checks the condition, then executes the body.
2. **for loop** – firstly initializes, then, condition check, execute body, update.
3. **do-while loop** – firstly, execute the body then condition check

for Loop

A for loop is a repetition control structure that allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

Syntax:

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

Example1:

```
for(int i = 0; i < n; i++)
{
    // BODY
}
```

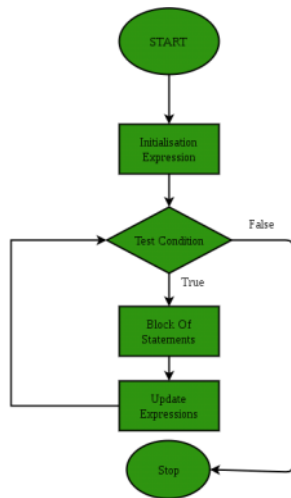
Example2:

```
for(auto element:arr)
{
    //BODY
}
```

In for loop, a loop variable is used to control the loop. First, initialize this loop variable to some value, then check whether this variable is less than or greater than the counter value. If the statement is true, then the loop body is executed and the loop variable gets updated. Steps are repeated till the exit condition comes.

- **Initialization Expression:** In this expression, we have to initialize the loop counter to some value. for example: `int i=1;`
- **Test Expression:** In this expression, we have to test the condition. If the condition evaluates to true then we will execute the body of the loop and go to update expression otherwise we will exit from the for a loop. For example: `i <= 10;`
- **Update Expression:** After executing the loop body this expression increments/decrements the loop variable by some value. for example: `i++;`

Flow Diagram of for loop:



Example1:

- C++

```
// C++ program to Demonstrate for loop
#include <iostream>
using namespace std;

int main()
{
    for (int i = 1; i <= 5; i++) {
        cout << "Hello World\n";
    }

    return 0;
}
```

Output

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Example2:

- C++

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

int main() {

int arr[] {40, 50, 60, 70, 80, 90, 100};
    for (auto element: arr){
        cout << element << " ";
    }
    return 0;

}
```

Output

```
40 50 60 70 80 90 100
```

While Loop

While studying **for loop** we have seen that the number of iterations is **known beforehand**, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where **we do not know** the exact number of iterations of the loop **beforehand**. The loop execution is terminated on the basis of the test conditions.

We have already stated that a loop mainly consists of three statements – initialization expression, test expression, and update expression. The syntax of the three loops – For, while, and do while mainly differs in the placement of these three statements.

Syntax:

```
initialization expression;
while (test_expression)
{
```

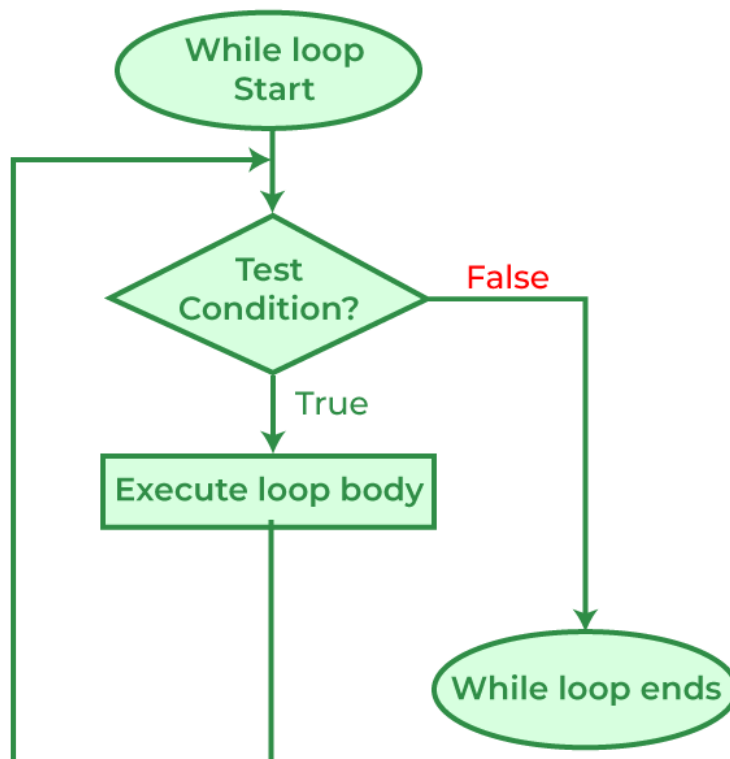
```

// statements

update_expression;
}

```

Flow Diagram of while loop:



Example:

- C++

```

// C++ program to Demonstrate while loop
#include <iostream>
using namespace std;

int main()
{
    // initialization expression
    int i = 1;

    // test expression
    while (i < 6) {
        cout << "Hello World\n";

        // update expression
        i++;
    }

    return 0;
}

```

Output

```

Hello World
Hello World
Hello World
Hello World
Hello World

```

do-while loop

In do-while loops also the loop execution is terminated on the basis of test conditions. The main difference between a do-while loop and the while loop is in the do-while loop the condition is tested at the end of the loop body, i.e do-while loop is exit controlled whereas the other two loops are entry-controlled loops.

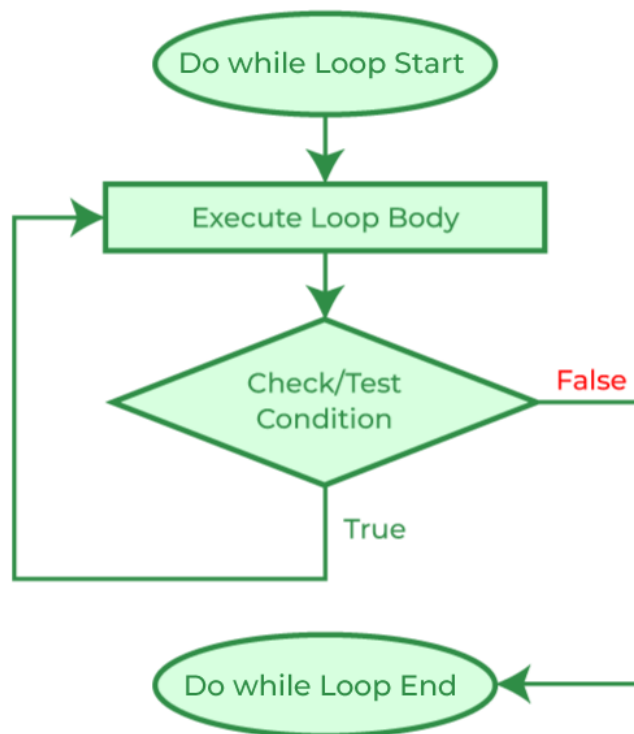
Note: In a do-while loop, the loop body will **execute at least once** irrespective of the test condition.

Syntax:

```
initialization expression;  
do  
{  
    // statements  
update_expression;  
} while (test_expression);
```

Note: Notice the semi – colon(“;”)in the end of loop.

Flow Diagram of the do-while loop:



Example:

- C++

```
// C++ program to Demonstrate do-while loop  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int i = 2; // Initialization expression  
  
    do {  
        // loop body  
        cout << "Hello World\n";  
    }
```

```

        // update expression
        i++;

    } while (i < 1); // test expression

    return 0;
}

```

Output

Hello World

In the above program, the test condition ($i < 1$) evaluates to false. But still, as the loop is an exit – controlled the loop body will execute once.

What about an Infinite Loop?

An infinite loop (sometimes called an endless loop) is a piece of coding that lacks a **functional exit** so that it repeats indefinitely. An infinite loop occurs when a condition is always evaluated to be true. Usually, this is an error.

Using For loop:

- C++

```

// C++ program to demonstrate infinite loops
// using for and while loop

// Uncomment the sections to see the output
#include <iostream>
using namespace std;
int main()
{
    int i;

    // This is an infinite for loop as the condition
    // expression is blank
    for (;;) {
        cout << "This loop will run forever.\n";
    }

    // This is an infinite for loop as the condition
    // given in while loop will keep repeating infinitely
    /*
    while (i != 0)
    {
        i-- ;
        cout << "This loop will run forever.\n";
    }
    */

    // This is an infinite for loop as the condition
    // given in while loop is "true"
    /*
    while (true)
    {
        cout << "This loop will run forever.\n";
    }
    */
}

```

Output:

```

This loop will run forever.
This loop will run forever.
.....

```

Using While loop:

- C++

```

#include <iostream>
using namespace std;

int main()

```

```

{
    while (1)
        cout << "This loop will run forever.\n";
    return 0;
}

```

Output:

```

This loop will run forever.
This loop will run forever.
.....

```

Using the Do-While loop:

- C++

```

#include <iostream>
using namespace std;

int main()
{
    do {
        cout << "This loop will run forever.\n";
    } while (1);

    return 0;
}

```

Output:

```

This loop will run forever.
This loop will run forever.
.....

```

C++ break-continue Statements

01 December 2022 17:08

C++ break

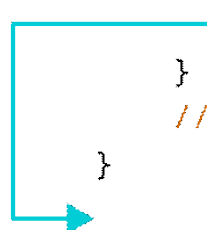
In C++, the break statement terminates the loop when it is encountered.

The syntax of the break statement is:

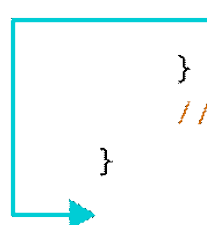
```
break;
```

Working of C++ break Statement

```
for (init; condition; update) {  
    // code  
    if (condition to break) {  
        break;  
    }  
    // code  
}
```



```
while (condition) {  
    // code  
    if (condition to break) {  
        break;  
    }  
    // code  
}
```



Working of break statement in C++

Example 1: break with for loop

```
// program to print the value of i#include<iostream>  
using namespace std;
```



```
int main(){
    for(int i = 1; i <= 5; i++) {
        // break condition    if(i == 3) {
            break;
        }
        cout<< i << endl;
    }
    return 0;
}
```

Output

```
1
2
```

In the above program, the for loop is used to print the value of i in each iteration. Here, notice the code:

```
if(i == 3) {
    break;
}
```

This means, when i is equal to **3**, the break statement terminates the loop. Hence, the output doesn't include values greater than or equal to 3.

Note: The break statement is usually used with decision-making statements.

Example 2: break with while loop

```
// program to find the sum of positive numbers// if the
// user enters a negative numbers, break ends the loop//
// the negative number entered is not added to sum
#include<iostream>
using namespace std;
int main(){
    int number;
    int sum = 0;
    while(true) {
```

```

    // take input from the user
    cout<< "Enter a number: ";
    cin>> number;
    // break condition
    if(number < 0) {
        break;
    }
    // add all positive numbers
    sum += number;
}
// display the sum
cout<< "The sum is "<< sum << endl;
return 0;
}

```

Output

```

Enter a number: 1
Enter a number: 2
Enter a number: 3
Enter a number: -5
The sum is 6.

```

In the above program, the user enters a number. The while loop is used to print the total sum of numbers entered by the user. Here, notice the code,

```

if(number < 0) {
    break;
}

```

This means, when the user enters a negative number, the break statement terminates the loop and codes outside the loop are executed.

The while loop continues until the user enters a negative number.

break with Nested loop

When break is used with nested loops, break terminates the inner loop. For example,

```
// using break statement inside// nested for loop
#include<iostream>
using namespace std;
int main(){
    int number;
    int sum = 0;
    // nested for loops// first loop
    for(int i = 1; i <= 3; i++) {
        // second loop
        for(int j = 1; j <= 3; j++) {
            if(i == 2) {
                break;
            }
            cout<< "i = "<< i << ", j = "<< j << endl;
        }
    }
    return 0;
}
```

Output

```
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

In the above program, the break statement is executed when $i == 2$. It terminates the inner loop, and the control flow of the program moves to the outer loop.

Hence, the value of $i = 2$ is never displayed in the output.

C++ loop exercises

01 December 2022 17:13

1. Program to find the length of a number in C++ (CPP). Program to find the square root of a number in C++.
2. Factorial Program in C, C++ (C Plus Plus, CPP) with flow chart.
3. Fibonacci Series Program in C++ and C with flowchart.
4. Palindrome Program in C, C++, C Plus Plus (CPP) with flow chart.
5. Program to Print sum of the number of Series in C Plus Plus (CPP, C++) and C with Flowchart.
6. C++ and C Program to Find the Number of Vowels, Digits, Consonants and White Spaces in a String with flowchart.
7. Program in C++, C to display the reverse of a number with a flowchart.
8. Write a program that lets the user enter the total rainfall for each of 12 months
9. Write a C++ program to print a triangle of prime numbers up to given number of lines of the triangle
10. Write a C++ program to print all numbers between a and b (a and b inclusive) using for loops

Nested for loop

Nested for loop means one or many for loops inside some another for loop.

11. Nested For Loop including 3 loops in C++ (C Plus Plus).
12. 3 Level Nested for loop in C++ (C Plus Plus).
13. Compound Assignment Operators in C++.
14. C++ program to show the sum of main diagonal elements by using nested for loop.
15. Write nested loops to print a rectangle. sample output for given program: * * * * *.
16. Write a C++ program to print the lower and upper triangles of a square matrix
17. Write a C++ program for trapezoidal rule

while loop

The while loop is a methodology to use a piece of code again and again until the given condition remains true. Loop will terminate when the given condition will false.

Just like for loop, the while loop has three parts. 1. Loop initialization, 2. Condition, and 3. increment or decrement of a loop.

18. Program of the sum of all digits of a number in C, C Plus Plus (CPP, C++) with flow chart.
- the
19. converts a number into binary in C++ and C with a flowchart.
20. Program in C Plus Plus and C with an explanation to find the Greatest Common Division (GCD). Solution with flowchart.
21. C++ Program to convert Octal to decimal number using while loop.
22. Write a program using integers usernum and x as input, and output usernum divided by x four times.
23. Write a C++ program to add two numbers without using the addition operator
24. Write a C++ program that simulates the rolling of two dice

Do while loop

In Do, while loop, first one time we execute the do part of the loop, and then we check the condition. It means that if the condition is false then still do part execute one time.

25. C++ Program to Display English Alphabets from A-Z.
26. Write a program in C++ to display the multiplication table vertically from 1 to n.
27. C++ Program to Find the Sum of Digits of a Number – C++ Program.
28. Program in C++ to display the n terms of even natural number and their sum
29. C++ program to find the sum of all integer between 100 and 200 which are divisible by 9.
30. Write a program in C++ to print Floyd's Triangle.
31. C++ program to print pyramid pattern of numbers.
32. C++ Program to display the n terms of odd natural number and their sum.
33. Convert a binary number to octal in C++.
34. c++ program to check whether a given number is a perfect number or not.
35. C++ program to find HCF (Highest Common Factor) of two numbers.

36. Program in C++ to convert a decimal number into binary without using an array.
37. Program C++ to convert an octal number into binary.
38. Write a c program to find out the sum of an A.P. series.
39. Program in C++, C to display the reverse of a number with flowchart.
40. Write the Octal to Decimal number program in C++.
41. C++ Program right angle triangle with a number that will repeat a number in a row.
42. C++ program to print rhombus or parallelogram star pattern.
43. Program in C++ to make such a pattern like a pyramid with a number that will repeat the number in the same row.
44. Write a program in C++ to Check Whether a Number can be Express as a Sum of Two Prime Numbers.
45. C++ program to print mirrored right triangle star pattern.
46. Decimal to octal number conversion program in C++.
47. C++ program to print a hollow square or rectangle star pattern.
48. C++ Program Sum of odd Natural Number.
49. The sum of the first 10 natural numbers program in C++.

C++ loop exercises

01 December 2022 17:13

1. Program to find the length of a number in C++ (CPP). Program to find the square root of a number in C++.
2. Factorial Program in C, C++ (C Plus Plus, CPP) with flow chart.
3. Fibonacci Series Program in C++ and C with flowchart.
4. Palindrome Program in C, C++, C Plus Plus (CPP) with flow chart.
5. Program to Print sum of the number of Series in C Plus Plus (CPP, C++) and C with Flowchart.
6. C++ and C Program to Find the Number of Vowels, Digits, Consonants and White Spaces in a String with flowchart.
7. Program in C++, C to display the reverse of a number with a flowchart.
8. Write a program that lets the user enter the total rainfall for each of 12 months
9. Write a C++ program to print a triangle of prime numbers up to given number of lines of the triangle
10. Write a C++ program to print all numbers between a and b (a and b inclusive) using for loops

Nested for loop

Nested for loop means one or many for loops inside some another for loop.

11. Nested For Loop including 3 loops in C++ (C Plus Plus).
12. 3 Level Nested for loop in C++ (C Plus Plus).
13. Compound Assignment Operators in C++.
14. C++ program to show the sum of main diagonal elements by using nested for loop.
15. Write nested loops to print a rectangle. sample output for given program: *
* * * * *
16. Write a C++ program to print the lower and upper triangles of a square matrix
17. Write a C++ program for trapezoidal rule

while loop

The while loop is a methodology to use a piece of code again and again until the given condition remains true. Loop will terminate when the given condition will false. Just like for loop, the while loop has three parts. 1. Loop initialization, 2. Condition, and 3. increment or decrement of a loop.

18. Program of the sum of all digits of a number in C, C Plus Plus (CPP, C++) with flow chart. the
19. converts a number into binary in C++and C with a flowchart.
20. Program in C Plus Plus and C with an explanation to find the Greatest Common Division (GCD). Solution with flowchart.
21. C++ Program to convert Octal to decimal number using while loop.
22. Write a program using integers usernum and x as input, and output usernum divided by x four times.
23. Write a C++ program to add two numbers without using the addition operator
24. Write a C++ program that simulates the rolling of two dice

Do while loop

In Do, while loop, first one time we execute the do part of the loop, and then we check the condition. It means that if the condition is false then still do part execute one time.

25. C++ Program to Display English Alphabets from A-Z.
26. Write a program in C++ to display the multiplication table vertically from 1 to n.
27. C++ Program to Find the Sum of Digits of a Number – C++ Program.
28. Program in C++ to display the n terms of even natural number and their sum
29. C++ program to find the sum of all integer between 100 and 200 which are divisible by 9.
30. Write a program in C++ to print Floyd's Triangle.
31. C++ program to print pyramid pattern of numbers.
32. C++ Program to display the n terms of odd natural number and their sum.
33. Convert a binary number to octal in C++.

34. c++ program to check whether a given number is a perfect number or not.
35. C++ program to find HCF (Highest Common Factor) of two numbers.
36. Program in C++ to convert a decimal number into binary without using an array.
37. Program C++ to convert an octal number into binary.
38. Write a c program to find out the sum of an A.P. series.
39. Program in C++, C to display the reverse of a number with flowchart.
40. Write the Octal to Decimal number program in C++.
41. C++ Program right angle triangle with a number that will repeat a number in a row.
42. C++ program to print rhombus or parallelogram star pattern.
43. Program in C++ to make such a pattern like a pyramid with a number that will repeat the number in the same row.
44. Write a program in C++ to Check Whether a Number can be Express as a Sum of Two Prime Numbers.
45. C++ program to print mirrored right triangle star pattern.
46. Decimal to octal number conversion program in C++.
47. C++ program to print a hollow square or rectangle star pattern.
48. C++ Program Sum of odd Natural Number.
49. The sum of the first 10 natural numbers program in C++.