

# **OBJECT ORIENTED PROGRAMMING LAB**



**Lab Manual # 03**

**Conditional Statements and Loops in C++**

**Instructor: Hurmat Hidayat**

**Semester Spring, 2022**

**Course Code: CL1004**

**Fast National University of Computer and Emerging Sciences  
Peshawar**

**Department of Computer Science**

# OBJECT ORIENTED PROGRAMMING LANGUAGE

## Table of Contents

Conditional Statements.....	1
1) if Statement .....	2
2) if else Statement .....	3
3. if-else-if else Statement .....	4
4. Nested if statement .....	8
5. Conditional Operator ( ? : ) .....	9
6. Switch statement .....	11
Control Structure/Loop .....	16
1. for loop .....	16
infinitive for loop.....	19
2. while loop.....	20
infinitive while loop.....	22
3. do while loop.....	22
infinitive do while loop.....	24
4. Enhanced for loop (for-each loop ) .....	25
Break Statement .....	26
Continue Statement .....	26
References .....	27

## Conditional Statements

### Relational Operators/comparison operators

Algebraic	In C++	Example	Meaning
>	>	$x > y$	x is greater than y
<	<	$x < y$	x is less than y
$\geq$	$\geq$	$x \geq y$	x is greater than or equal to y
$\leq$	$\leq$	$x \leq y$	x is less than or equal to y
=	==	$x == y$	x is equal to y
$\neq$	!=	$x != y$	x is not equal to y

### Blocks of Code

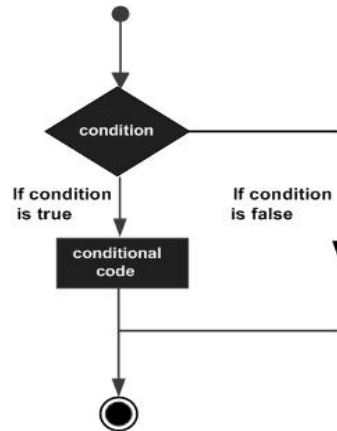
Whenever we write an if statement or a loop, if there is more than one statement of code which has to be executed, this has to be enclosed in braces, i.e. '{ .... }'

### Conditional Statements

- ❖ Also called decision making statements decision control statements.
- ❖ Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- ❖ Used to change the flow.
- ❖ In these statements (conditions) order or sequence of the statements are changed.
- ❖ Sometimes we need to execute a block of statements only when a particular condition is met or not met. This is called **decision making**, as we are executing a certain code after making a decision in the program logic.

1. If Statement
2. If-else Statement
3. If-else if- else statement
4. Switch statement
5. Nested Statement

- Use **if** to specify a block of code to be executed, if a specified condition is true.
- Use **else** to specify a block of code to be executed, if the same condition is false.
- Use **else if** to specify a new condition to test, if the first condition is false.
- Use **switch** to specify many alternative blocks of code to be executed.



## 1) if Statement

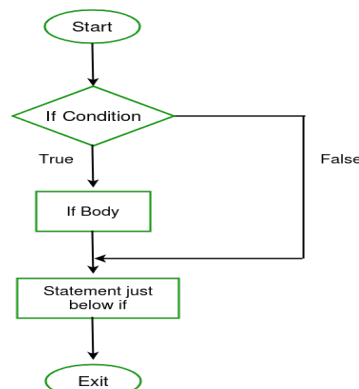
**if statement** will execute or skip or ignore a block of code depending one condition

**Syntax:**

```

if(conditional expression)
{
    // Statements will execute if the Boolean expression is true
}
  
```

The statements inside **if** parenthesis (usually referred as if body) gets executed only when the given condition is true. If the condition is false than the statements inside if body are completely ignored.



```
#include<iostream>
using namespace std;
int main()
{
    if (20>18)
    {
        cout<<"20 is greater than 18";
    }
}

/*
Output
20 is greater than 18
*/
```

**We can also test variables:**

```
#include <iostream>
using namespace std;
int main() {
    int x = 20;
    int y = 18;
    if (x > y) {
        cout << "x is greater than y";
    }
    return 0;
}

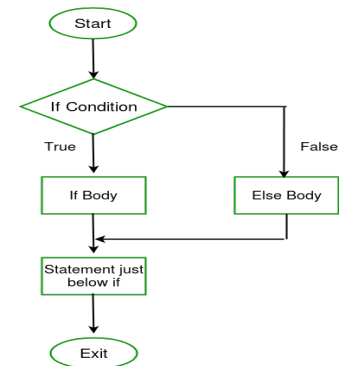
/*
Output
20 is greater than 18
*/
```

## 2) if else Statement

- ❖ Used for making two way decision.
- ❖ It will execute if block if condition is true and will execute another block (else block) if condition is false.
- ❖ It will take one action if the condition is true and take another action if condition is false.

- ❖ Sometimes you have a condition and you want to execute a block of code if condition is true and execute another piece of code if the same condition is false. This can be achieved in C++ using if-else statement.
- ❖ This is how an if-else statement looks:

```
if(condition)
{
    Statement(s);
}
else
{
    Statement(s);
}
```



The statements inside “if” would execute if the condition is true, and the statements inside “else” would execute if the condition is false.

```
#include<iostream>
using namespace std;
int main()
{   int x=15;
    int y=18;

    if (x>y)
    {
        cout<<"x is greater than y";
    }
    else
    {
        cout<<"y is greater than x";
    }
    return 0;
}

/*
Output
y is greater than x
*/
```

### 3. if-else-if else Statement

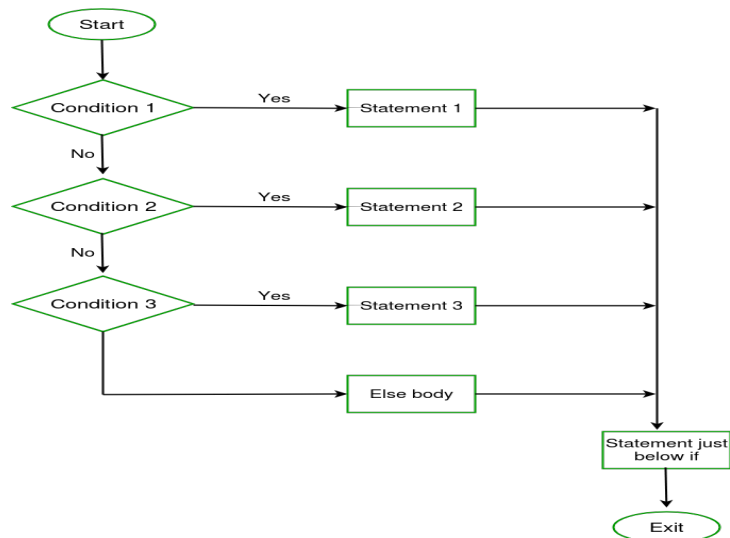
- ❖ This statement is used to check multiple conditions.

- ❖ Used to execute one condition from multiple statements.
- ❖ if-else-if statement is used when we need to check multiple conditions. In this control structure we have only one “if” and one “else”, however we can have multiple “else if” blocks. This is how it looks:

```
if(condition_1)
{
/*if condition_1 is true execute this*/
statement(s);
}
else if(condition_2)
{
/*execute this if condition_1 is not met and condition_2 is met*/
statement(s);
}
else if(condition_3)
{ /* execute this if condition_1 & condition_2 are not met and
condition_3 is met */
statement(s);
}
.
.
.
else {
/* if none of the condition is true then these statements gets
executed */
statement(s);
}
```

**Note:**

- ❖ The most important point to note here is that in if-else-if, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored.
- ❖ If none of the condition is met then the statements inside “else” gets executed.



```

#include<iostream>
using namespace std;

int main()
{
    int x=20;

    if (x==10)
    {
        cout<<"x is 10";
    }
    else if(x==15)
    {
        cout<<"x is 15";
    }
    else if(x==20)
    {
        cout<<"x is 20";
    }
    else
    {
        cout<<"sorry! x is not present";
    }

    return 0;
}

/*
Output
x is 20
*/

```



```
#include<iostream>
using namespace std;
int main()
{
    int number;
    cout<<"Enter a number between 1 to 3:";
    cin>> number;
    if(number == 1)
    {
        cout<<"You pressed 1" <<endl;
    }
    else if(number == 2)
    {
        cout<<"You pressed 2" <<endl;
    }
    else if(number == 3)
    {
        cout<<"You pressed 3" <<endl;
    }
    else
    {
        cout<<"Invalid input";
    }
}

/*
Output
Enter a number between 1 to 3: 3
You pressed 3
Enter a number between 1 to 3: 50
Invalid input
*/
```

## 4. Nested if statement

You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

```
#include<iostream>
using namespace std;
int main()
{
    int x=30;
    int y=10;
    if (x==30)
    {
        if(y==10)
        {
            cout<<"x=30 and y=10";
        }
    }
    return 0;
}

/*
Output
x=30 and y=10
*/
```

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

int main()
{
    bool job; char martialStatus; int age;
    cout<<"Enter Martial Status:";
    cin>>martialStatus;
    cout<<"Enter age:";
    cin>>age;
    cout<<"Enter job Status:";
    cin>>job;
    if(martialStatus == 'u')
    {
        if(age <=25)
            if(job ==false)
                cout<<"Eligible for Loan";
    }

    else
```

```
        cout<<"Not eligible for Loan";  
        return 0;  
    }
```

```
/*
```

**Output**

```
Enter Marital Status: u  
Enter age:20  
Enter job Status: 0  
Eligible for Loan
```

```
Enter Marital Status: u  
Enter age:18  
Enter job Status: 1  
Not eligible for Loan
```

```
*/
```

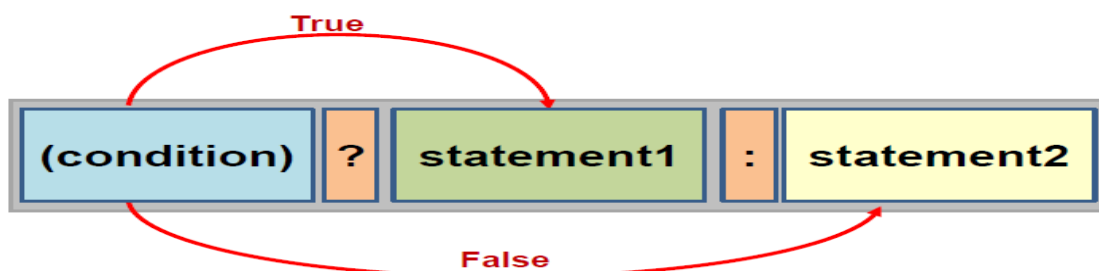
## 5. Conditional Operator (? :)

It is ternary operator and work on three operands. It works like if else statement.

**Syntax:**

(condition) ? statement 1 : statement 2;

Statement must be single and it is the limitation of conditional operator.



### Conditional Operator (? :) Example

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

```
{
    int n1=5;
    int n2=4;
    int result = (n1>n2) ? n1 : n2;
    cout<<result;
return 0;

}

/*
Output
5
*/
```

## Logical Operators

Used for compound condition or expression

- ❖ AND (&&)
- ❖ OR (||) pip sign

### 1. AND (&&)

```
#include<iostream>
using namespace std;
int main()
{
    int x=30;
    int y=10;

    if (x==30 && y==10)
    {
        cout<<"x=30 and y=10";
    }
    else
    {
        cout<<"Either x is not equal to 30 or y is not equal to 10 or both
are not equal";
    }
return 0;
}

/*
Output
*/
```

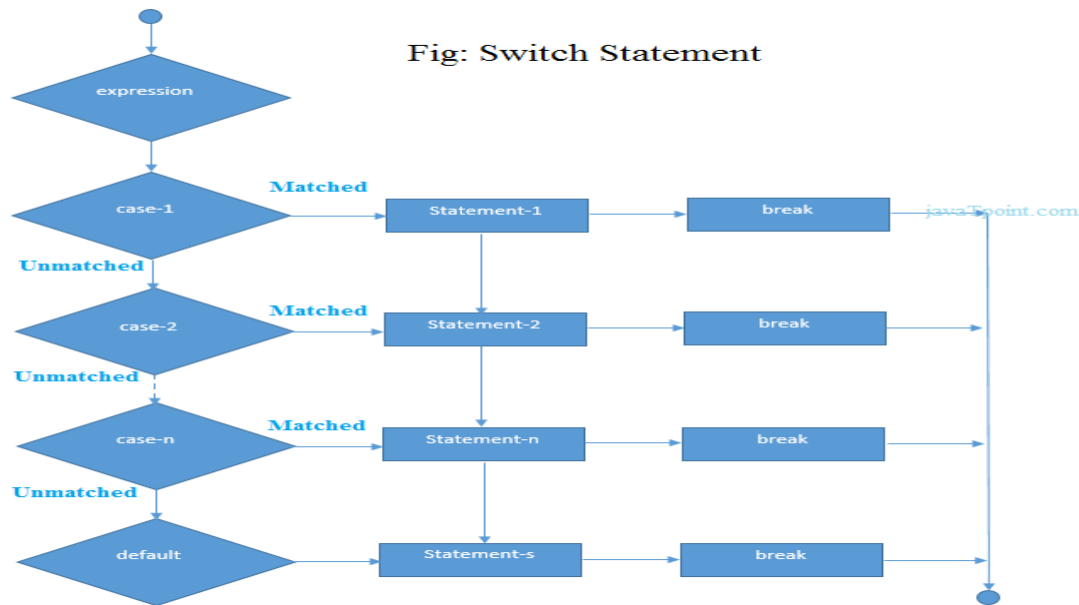
## 2. OR (||)

```
#include<iostream>
using namespace std;
int main()
{
    int x=30;
    int y=10;

    if (x==30 || y==10)
    {
        cout<<"x=30 or y=10";
    }
    else
    {
        cout<<"Both are not equal means x is not equal to 30 and y is not
equal to 10";
    }
    return 0;
}
```

## 6. Switch statement

- ❖ Used when multiple choices are given and one is to be selected. It is like if else-if- else statement.
- ❖ Used to select one several actions based on the value of variable or expression.
- ❖ **Switch case statement** is used when we have number of options (or choices) and we may need to perform a different task for each choice.



### Switch statement Syntax

```
switch(variable/expression)
{
case value 1:
statement(s); // code to be executed
break;
case value 2:
statement(s); // code to be executed
break;
.
.
.
case value n:
Statement(s); // code to be executed
break;
default:
statement(s); // code to be executed if all cases are not matched
}
```

The screenshot shows a C++ IDE with a file named 'main.cpp'. The code is as follows:

```

1  #include <iostream>
2  using namespace std;
3  int main() {
4
5      int i=2;
6      switch(i) {
7          case 1:
8              cout<<"Case1 " <<endl;
9              break;
10         case 2:
11             cout<<"Case2 " <<endl;
12             break;
13         case 3:
14             cout<<"Case3 " <<endl;
15             break;
16         case 4:
17             cout<<"Case4 " <<endl;
18             break;
19         default:
20             cout<<"Default " <<endl;
21     }
22
23     return 0;
24 }

```

To the right of the code editor, there is a terminal window titled 'C:\Users\abdu\OneDrive\Documents\Project2.exe'. It displays the output of the program:

```

Case2
-----
Process exited after 0.06225 seconds with return value 0
Press any key to continue . . .

```

### Switch statement VS if else if

- ❖ If a program contains conditions or compound conditions then we use if else if else. If program contains single variable or expression then we use switch statement.

#### Compound conditions

- 1) if(a>b && a>c)
- 2) if(a>b || a>c)

### C++ Switch statement is fall through

It means it executes all statements after match if break statement is not used with switch cases.

The screenshot shows a C++ IDE with a file named 'main.cpp'. The code is as follows:

```

1  #include <iostream>
2  using namespace std;
3  int main() {
4
5      int i=2;
6      switch(i) {
7          case 1:
8              cout<<"Case1 " <<endl;
9          case 2:
10             cout<<"Case2 " <<endl;
11          case 3:
12             cout<<"Case3 " <<endl;
13          case 4:
14             cout<<"Case4 " <<endl;
15          default:
16             cout<<"Default " <<endl;
17     }
18
19     return 0;
20 }

```

To the right of the code editor, there is a terminal window titled 'C:\Users\abdu\OneDrive\Documents\Project2.exe'. It displays the output of the program:

```

Case2
Case3
Case4
Default
-----

```

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Enter alphabet: ";
    char alphabet;
    cin>>alphabet;
    switch(alphabet)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U':
            cout<<"You entered vowel";
            break;
        default:
            cout<<"You entered consonant";
    } // switch body Closed
    return 0;
}

/*
Output
Enter alphabet: a
You entered vowel

Enter alphabet: I
You entered vowel

Enter alphabet: s
You entered consonant
*/
```

## Break Statement

The break statement is used to exit from the body of the switch structure or loop structure.



The break statement terminates the execution of the loop when it is used inside the body of the loop.

**Syntax:** break;

### **boolean variable**

**Syntax:**

bool variable\_name;

e.g: bool even;

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Enter any number: ";
    int number;
    cin>>number;
    bool even;    // bool variable declaration
    even =(number % 2==0);
    if(even)
    {
        cout<<"Even number";
    }
    else
    {
        cout<<"Odd Number";
    }
    return 0;
}
```

/\*

### **Output**

Enter any number: 6  
Even number

Enter any number: 7  
Odd Number

\*/

## Control Structure/Loop

- ❖ In computer programming, a loop is a sequence of instructions that is continually repeated until a certain condition is reached.
- ❖ A loop statement allows us to execute a statement or group of statements multiple times.
- ❖ Loops are used in programming to repeat a specific block until some end condition is met.

There are four type of loops in C++ programming:

- 1) for loop
- 2) while loop
- 3) do while loop
- 4) for-each loop (Enhanced for Loop)

**Pretested Loop:** Loop in which condition is checked first.

e.g. for loop and while loop

**Post tested Loop:** Loop in which condition is checked at the end.

e.g. do while loop

**Determined loop/Definite loop:** Loop for fixed repetition.

e.g. for loop

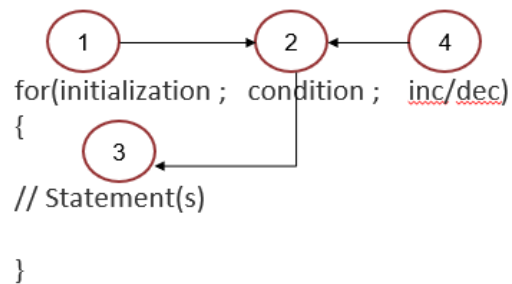
**Undetermined loop/Indefinite loop:** loop not for fixed repetition

E.g. while loop and do while loop

### 1. for loop

- ❖ for loop is used to a statement or group of statement for a fixed number of time.
- ❖ If the number of iterations is fixed then it is recommended to use for loop.
- ❖ A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

**Syntax:**



Order of Steps in for loop:

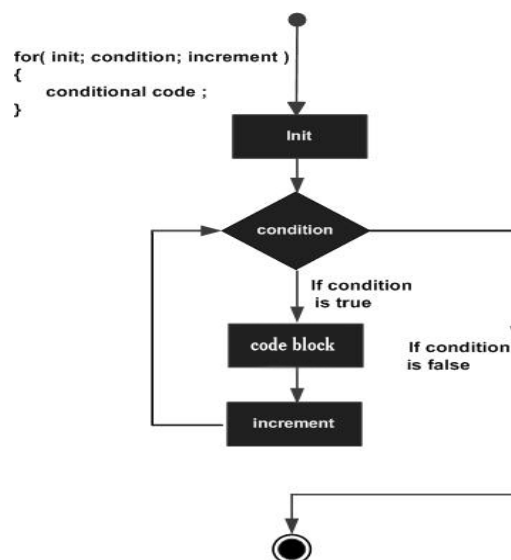
1. 1<sup>st</sup> initialization is performed.
2. Secondly condition is checked
3. In 3<sup>rd</sup> step statement is executed means control goes to body of the loop.
4. In 4<sup>th</sup> step increment or decrement is performed.
5. Again condition is checked and so on.

Note: In loops variable is called counter variable.

`i = i+1;`

OR

`i += 1;`



```

#include <iostream>
using namespace std;
int main() {
for (int i = 0 ; i< 5 ; i++)
{
    cout << i << "\n";
}
return 0;
}
/*

```

**Output**

```
0
1
2
3
4
*/
```

```
#include <iostream>
using namespace std;
int main ()
{
    // for loop execution
    for( int a = 10 ; a < 20; a++ )
    {
        cout << "value of a: " << a << endl ;
    }
    return 0;
}

/*
output
value of a:10
value of a:11
value of a:12
value of a:13
value of a:14
value of a:15
value of a:16
value of a:17
value of a:18
value of a:19
*/
```

```
#include <iostream>
using namespace std;
int main ()
{
    for( int a = 0; a < =10; a++ )
    {
        cout << "value of a: " << a << endl;
    }
    return 0;
```

```
}

/*
Output
value of a: 0
value of a: 1
value of a: 2
value of a: 3
value of a: 4
value of a: 5
value of a: 6
value of a: 7
value of a: 8
value of a: 9
Value of a: 10
*/
```

```
#include <iostream>
using namespace std;
int main() {
    for (int i = 0 ; i <= 10; i = i + 2)
    {
        cout << i << "\n";
    }
    return 0;
}

/*
Output
0
2
4
8
10
*/
```

### **infinite for loop**

If you use two semicolons (;;) in the for loop it will be infinite for loop.

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

int main () {
```

```

for (;;)
{
    cout<<"infinite for loop";
}

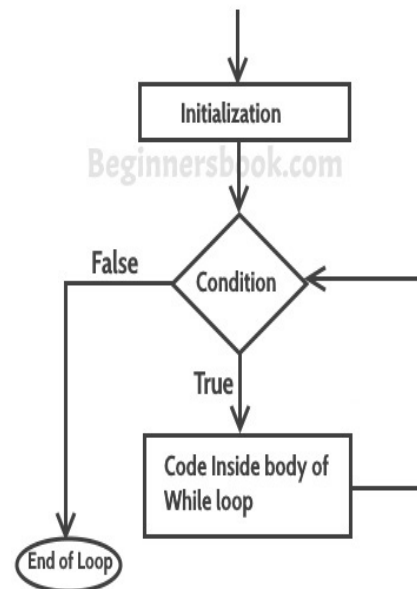
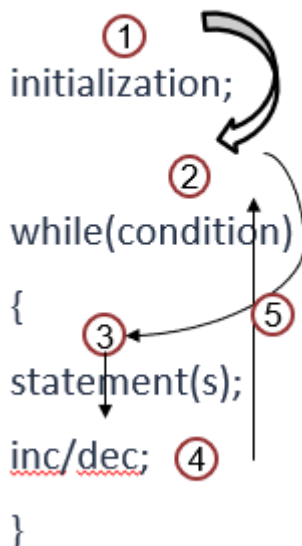
return 0;
}

```

## 2. while loop

- ❖ Is used when number of iterations is not fixed.
- ❖ A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

### Syntax



- Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.

```

#include <iostream>
using namespace std;
int main() {
    int i = 0;
    while (i < 5) {

        cout << i << "\n";
        i++;
    }
}

```

```
    }  
    return 0;  
}  
  
/*  
Output  
0  
1  
2  
3  
4  
*/
```

**Note:** Do not forget to increase the variable used in the condition, otherwise the loop will never end!

```
#include <iostream>  
using namespace std;  
  
int main () {  
    // Local variable declaration:  
    char c = 'n';  
    // while loop execution  
    while( c != 'y' )  
    {  
        cout << "in loop"<<endl;  
        cout<<"Exit while loop(y/n)?";  
        cin>>c;  
    }  
    cout<<"====OUTSIDE LOOP===="<<endl;  
    return 0;  
}  
  
/*  
Output  
*/
```

```

Select C:\Users\This Pc\documents\visual studio 2012\Projects\CSBlab\Debug\
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?y
====OUTSIDE LOOP====
Press any key to continue . . . _

```

**Sentinel Condition:** Truthfulness or falseness depends upon user input.

### infinite while loop

If you pass true or true value in the while loop, it will be infinite while loop.

#### Syntax:

```

while(true)
{
    Statement(s);
}

```

```

#include <iostream>
using namespace std;

int main () {
    while(true)
    {
        cout<<"infinite while loop";
    }

    return 0;
}

```

## 3. do while loop

An indefinite loop. Best used when the number of iteration is unknown. Used when you will execute the loop at least once. Unlike **for** and **while** loops, which test the loop condition at the top of the loop,



the **do...while** loop checks its condition at the bottom of the loop. A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

### Syntax

initialization;

do

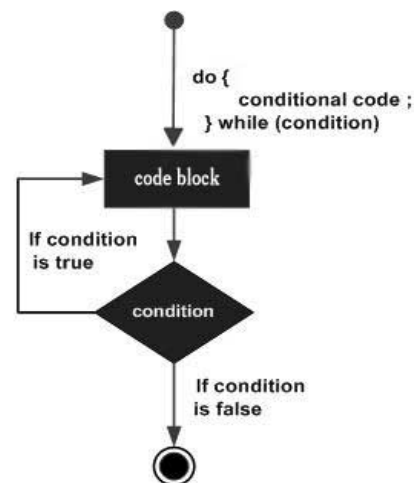
{

statement(s);

inc/dec;

}

while(condition) ;



- ❖ Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.
- ❖ If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

```

#include <iostream>
using namespace std;

int main() {
    int i = 0;
    do {
        cout << i << "\n";
        i++;
    }
    while (i < 5);
    return 0;
}

```

/\*

**Output**

0

2

3

4

\*/

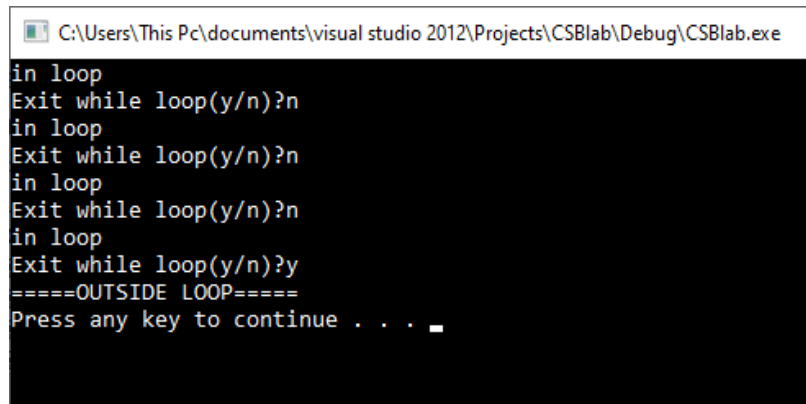
Do not forget to increase the variable used in the condition, otherwise the loop will never end!

```

#include <iostream>
using namespace std;
int main ()
{
    // Local variable declaration:
    char c ;
    // do-while loop execution
    do
    {
        cout << "in loop"<<endl;
        cout<<"Exit while loop(y/n)?";
        cin>>c;
    }
    while( c !='y' );

    cout<<"====OUTSIDE LOOP===="<<endl;
    return 0;
}

```



```

C:\Users\This Pc\documents\visual studio 2012\Projects\CSBlab\Debug\CSBlab.exe
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?y
====OUTSIDE LOOP====
Press any key to continue . . .

```

### infinite do while loop

If you pass true or true value in the do while loop, it will be infinite do while loop.

#### Syntax:

```

do
{
Statement(s);
}

```

```
while(true);
```

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

int main () {
    while(true)
    {
        cout<<"infinite do while loop";
    }

    return 0;
}
```

#### 4. Enhanced for loop (for-each loop )

- ❖ Works with array.
- ❖ Is used for traversing in array.
- ❖ It is easy to use than simple for loop because here we do not initialization of counter variable, condition and incrementation or decrementation of counter variable.

##### Syntax

```
for (datatype variable-name : arrayname)
```

```
{ statement(s); }
```

**Data type must be same as that of array data type**

```
#include<iostream>
using namespace std;
int main()
{
    int arr[]={1,2,3,4,5}; //array initialization
    cout<<"The elements are: ";
    for(int i : arr)
    {
        cout<<i<<" ";
    }
    return 0;
}

/*
```

**Output**

The elements are: 1 2 3 4 5  
\*/

## Break Statement

The break statement terminates the execution of the loop when it is used inside the body of the loop.

**Syntax:** break;

The break statement can also be used to jump out of a **loop**.

```
#include <iostream>
using namespace std;
int main() {
    for (int i = 0; i <10; i++)
    {
        if (i == 4) {
            break;
        }
        cout << i << "\n";
    }
    return 0;
}
```

/\*

**Output**

0  
1  
2  
3  
\*/

## Continue Statement

- ❖ The continue statement shifts the control back to the beginning of the loop.
- ❖ It is used inside the body of the loop.
- ❖ It is used to continue loop.
- ❖ It continues the current flow of the program and skips the remaining code at specified condition.

**Syntax:** continue;

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

int main() {
    for (int i = 0; i < 10; i++)
    {
        if (i == 4) {
            continue;
        }
        cout << i << "\n";
    }
    return 0;
}
```

```
/*
Output
0
1
2
3
5
6
7
8
9
*/
}
```

## References

<https://beginnersbook.com/2017/08/cpp-data-types/>

[http://www.cplusplus.com/doc/tutorial/basic\\_io/](http://www.cplusplus.com/doc/tutorial/basic_io/)

<https://www.w3schools.com/cpp/default.asp>

<https://www.javatpoint.com/cpp-tutorial>

<https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp>

<https://www.programiz.com/>

<https://ecomputernotes.com/cpp/>