MNNIT
COMPUTER CODING
CLUB

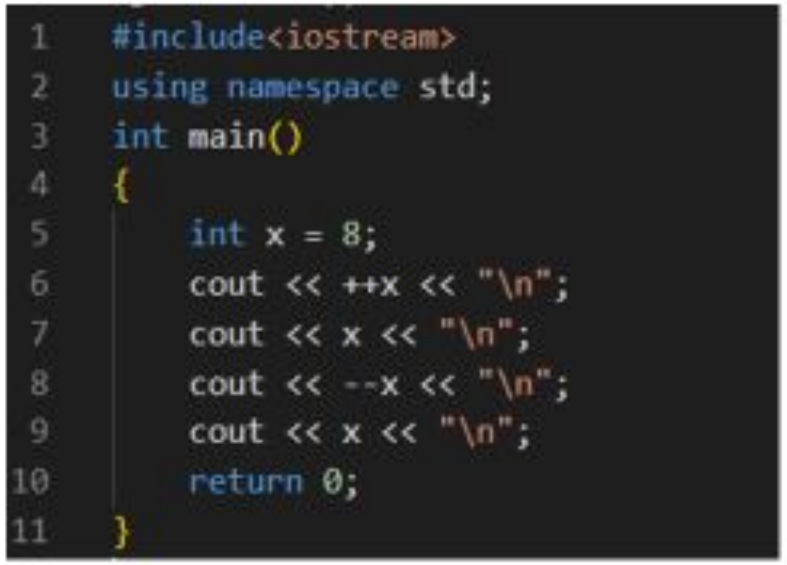
C L A S S - 2

Basics Of C++

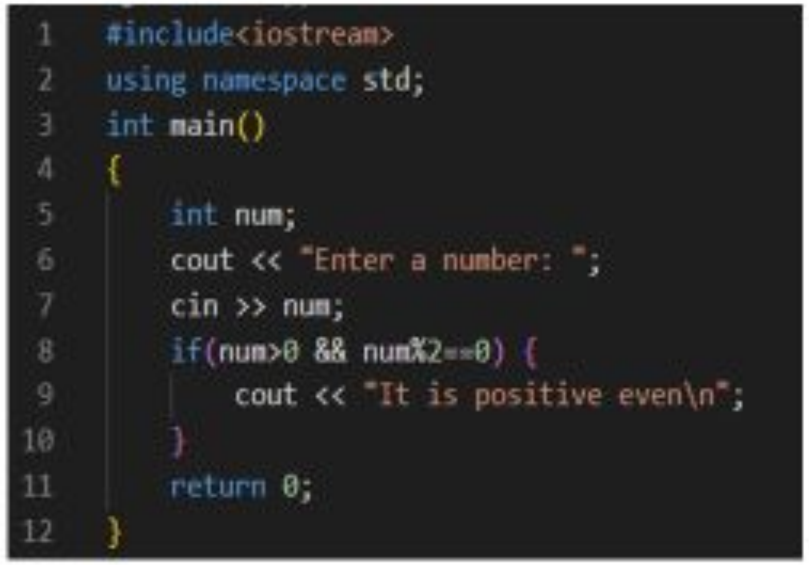
Quick Recap

Types of operators –

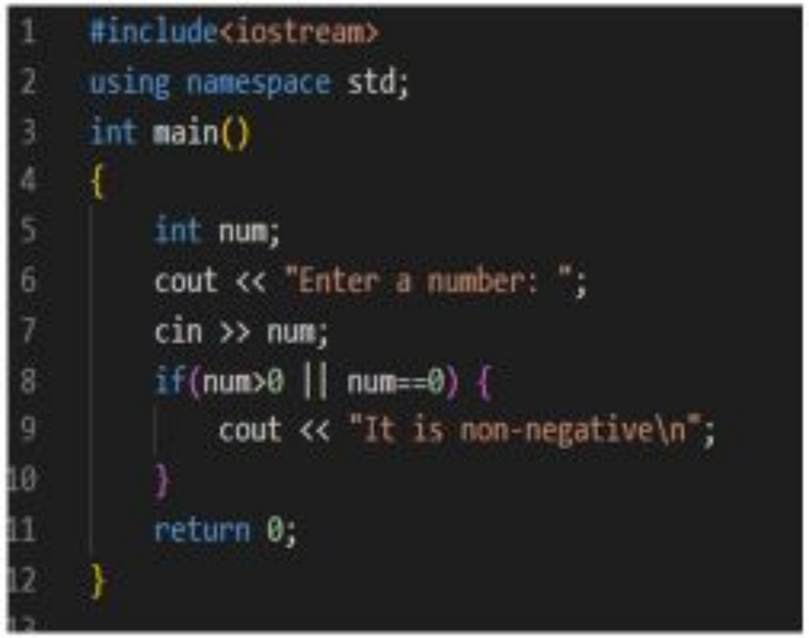
- Logical
- Arithmetic
- Bitwise
- Relational
- Assignment
- Conditional



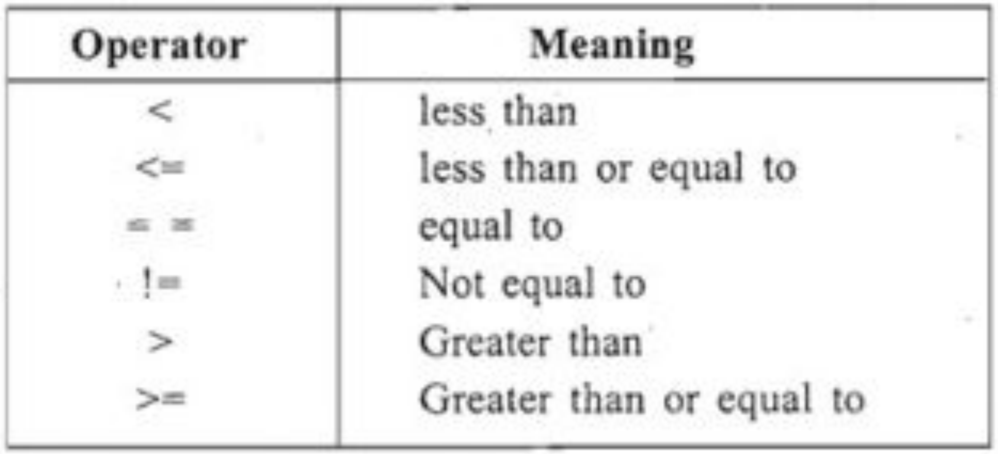
```cpp
#include<iostream>
using namespace std;
int main()
{
    int x = 8;
    cout << ++x << "\n";
    cout << x << "\n";
    cout << --x << "\n";
    cout << x << "\n";
    return 0;
}
```



```cpp
#include<iostream>
using namespace std;
int main()
{
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if(num>0 && num%2==0) {
        cout << "It is positive even\n";
    }
    return 0;
}
```

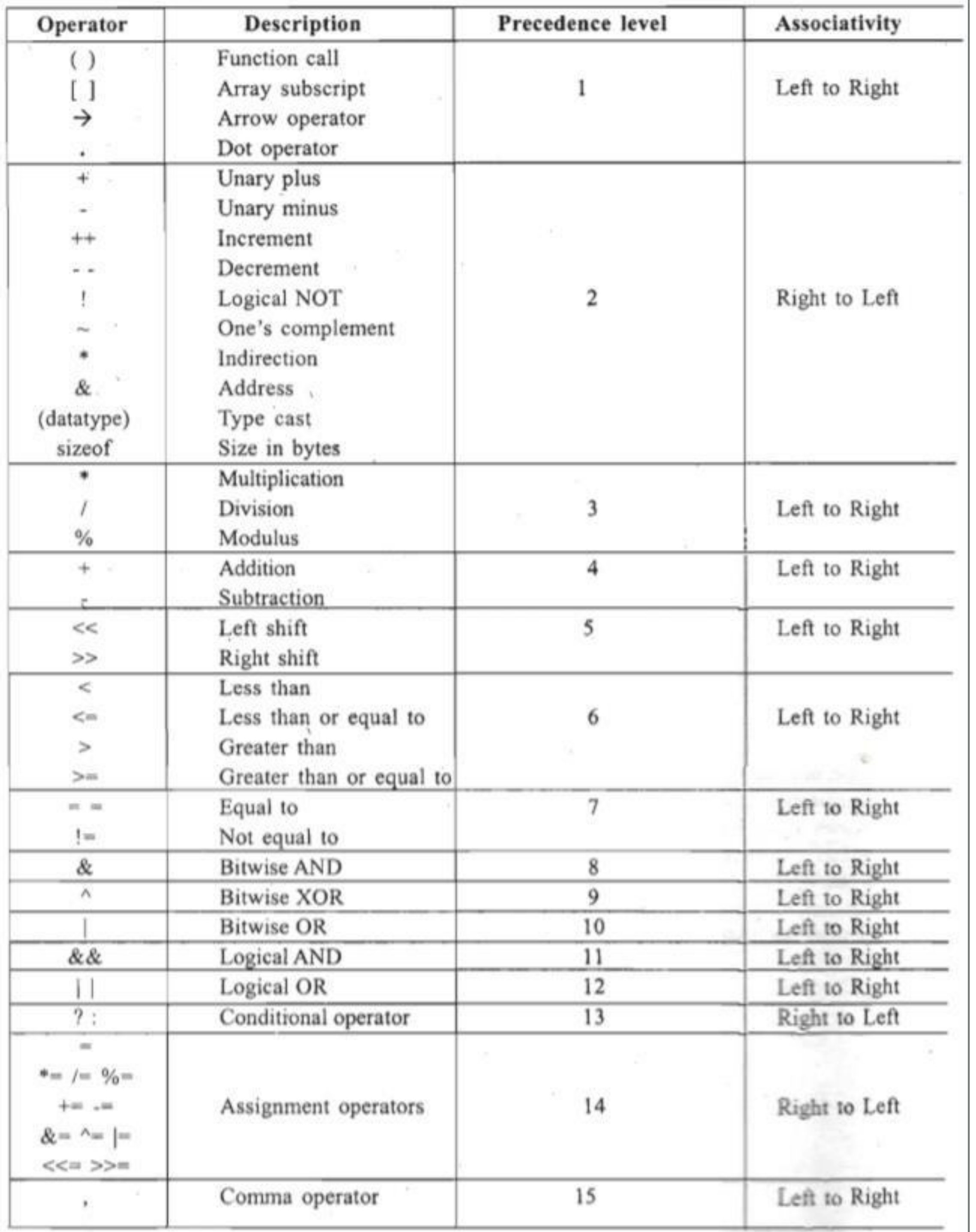


```cpp
#include<iostream>
using namespace std;
int main()
{
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if(num>0 || num==0) {
        cout << "It is non-negative\n";
    }
    return 0;
}
```



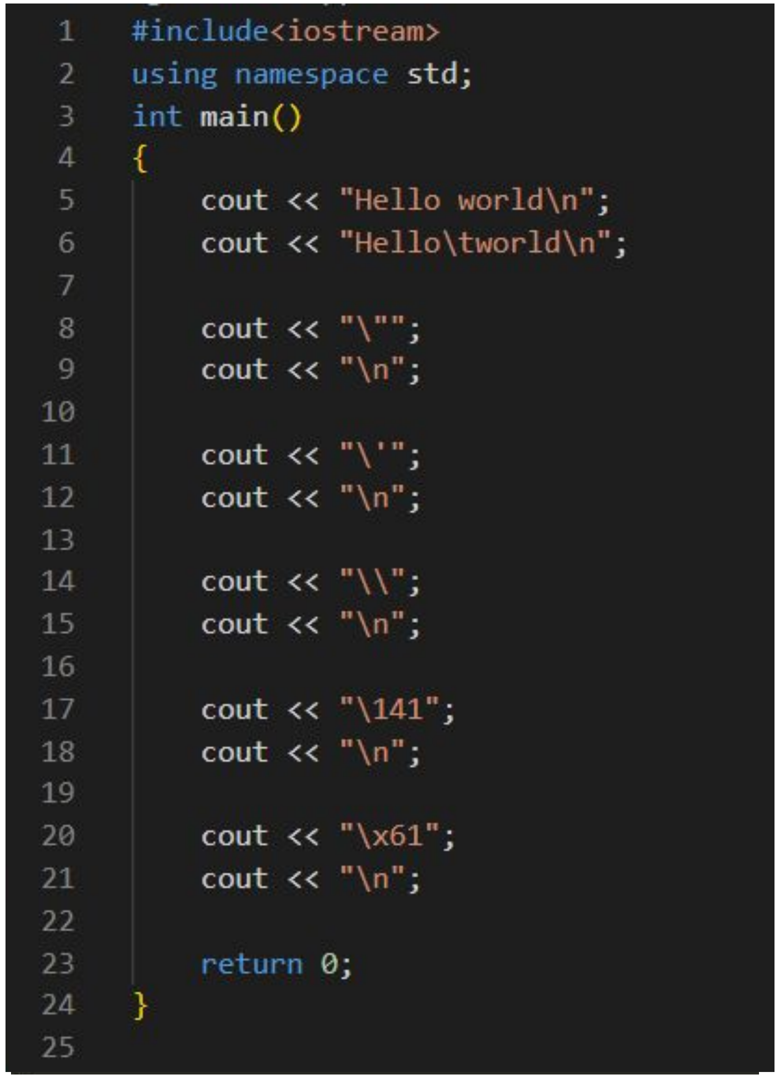
Operator	Meaning
<	less than
<=	less than or equal to
= =	equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to

PRECEDENCE AND ASSOCIATIVITY



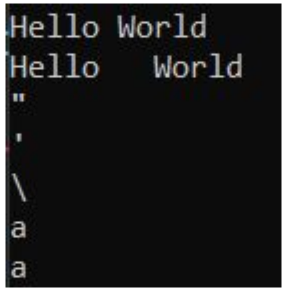
Operator	Description	Precedence level	Associativity
( )	Function call	1	Left to Right
[ ]	Array subscript		
→	Arrow operator		
.	Dot operator		
+	Unary plus	2	Right to Left
-	Unary minus		
++	Increment		
- -	Decrement		
!	Logical NOT		
~	One's complement		
*	Indirection		
&	Address		
(datatype)	Type cast		
sizeof	Size in bytes		
*	Multiplication	3	Left to Right
/	Division		
%	Modulus		
+	Addition	4	Left to Right
-	Subtraction		
<<	Left shift	5	Left to Right
>>	Right shift		
<	Less than	6	Left to Right
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
= =	Equal to	7	Left to Right
!=	Not equal to		
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
| |	Logical OR	12	Left to Right
? :	Conditional operator	13	Right to Left
=	Assignment operators	14	Right to Left
*= /= %=			
+= -=			
&= ^= |=			
<<= >>=			
,	Comma operator	15	Left to Right

ESCAPE SEQUENCES



```cpp
#include<iostream>
using namespace std;
int main()
{
    cout << "Hello world\n";
    cout << "Hello\tworld\n";

    cout << "\"";
    cout << "\n";

    cout << "\'";
    cout << "\n";

    cout << "\\";
    cout << "\n";

    cout << "\141";
    cout << "\n";

    cout << "\x61";
    cout << "\n";

    return 0;
}
```

Output:



```
Hello World
Hello   World
"
'
\
a
a
```

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

CONTROL STATEMENTS

The control flow statement in a language specify the order in which computations are performed.

As discussed in the previous class, expressions such as

- x = 0;
- i++;
- printf(...);

become a statement when followed by a semicolon.

Braces { and } are used to group declarations and statements together into a compound statement, or block, so that they are syntactically equivalent to a single statement.

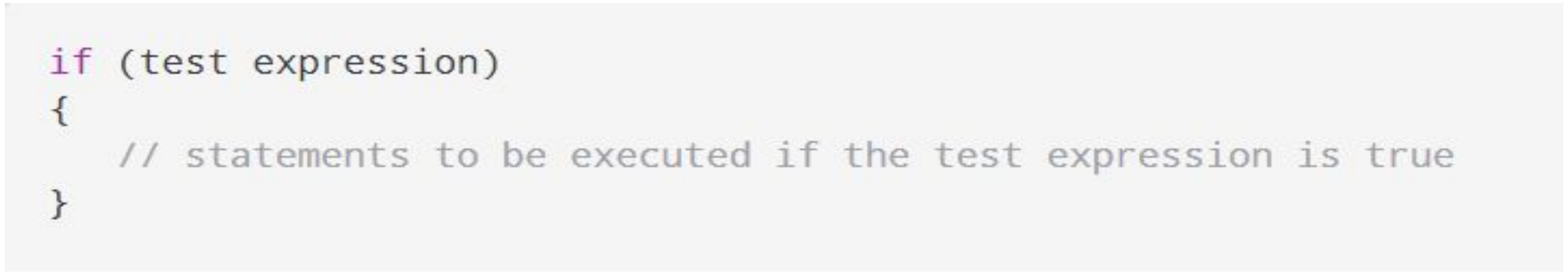
C++ provides two styles of flow control:

- Branching
- Looping

Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

IF STATEMENT

- The syntax of the `if` statement in C++ programming is:



```
if (test expression)
{
   // statements to be executed if the test expression is true
}
```

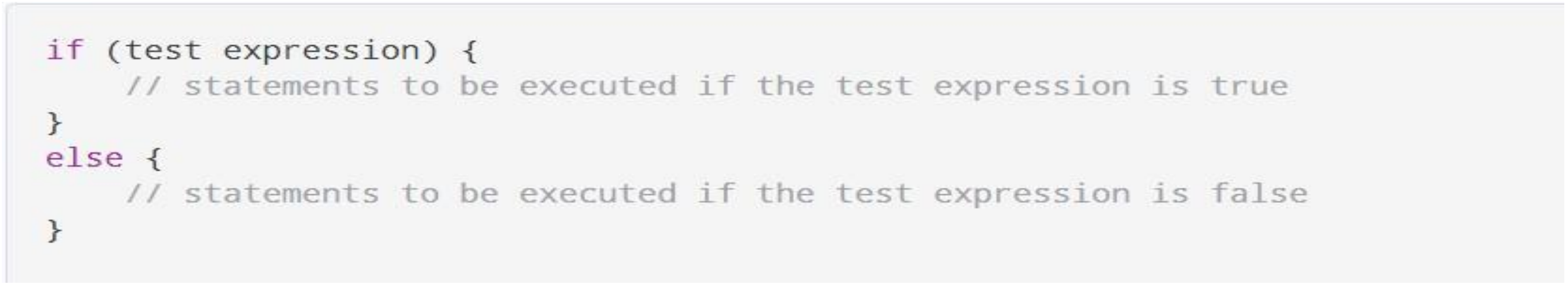
Working

The `if` statement evaluates the test expression inside the parenthesis `()`.

- If the test expression is evaluated to true, statements inside the body of `if` are executed.
- If the test expression is evaluated to false, statements inside the body of `if` are not executed.

IF ELSE STATEMENT

The `if` statement may have an optional `else` block. The syntax of the `if..else` statement is:

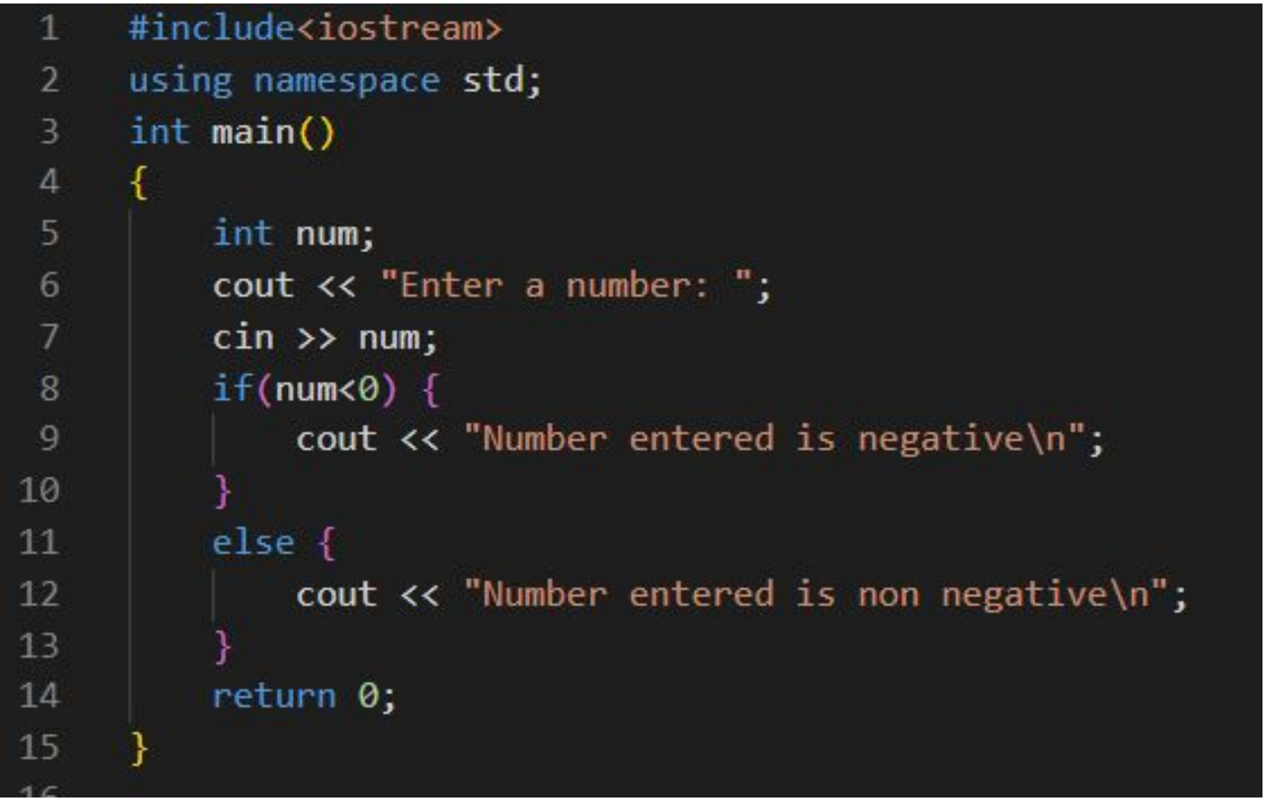


```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

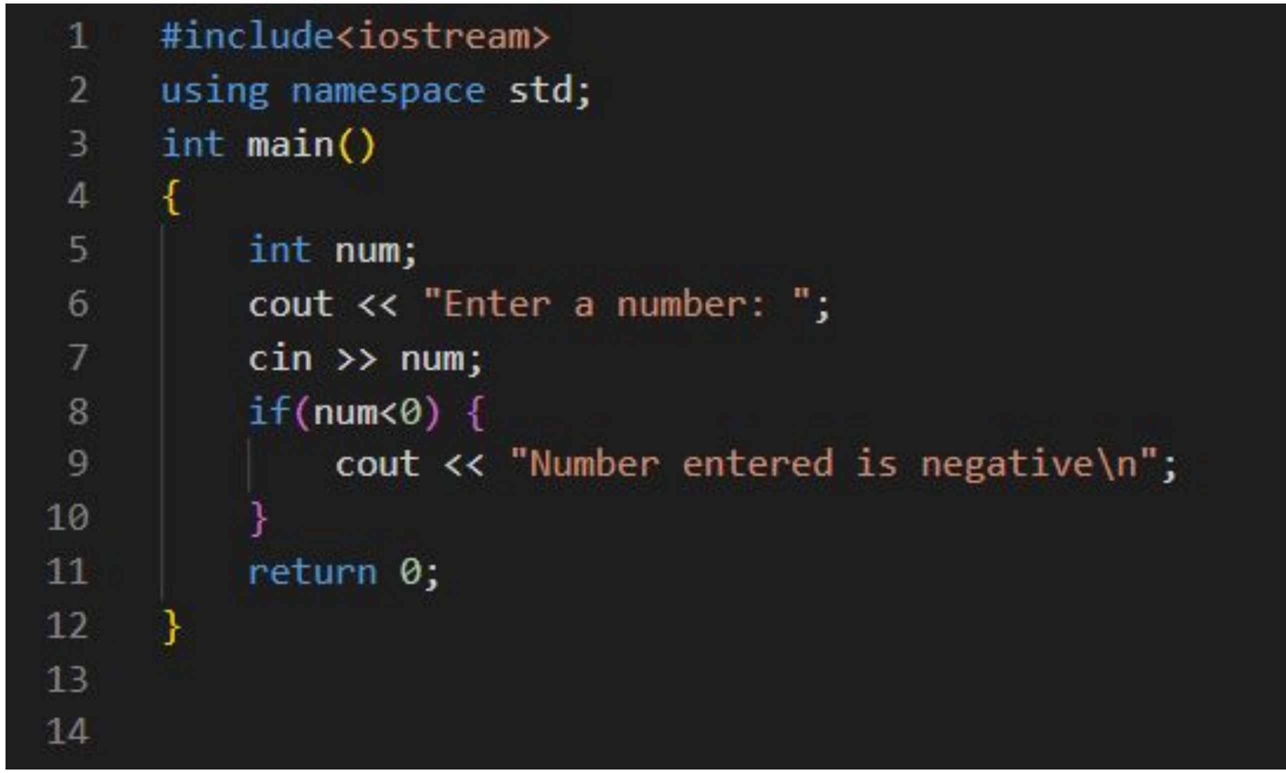
Working

If the test expression is evaluated to true,

- statements inside the body of `if` are executed.
- statements inside the body of `else` are skipped from execution. If the test expression is evaluated to false,
- statements inside the body of `else` are executed
- statements inside the body of `if` are skipped from execution.
- Conditional Operator



```cpp
#include<iostream>
using namespace std;
int main()
{
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if(num<0) {
        cout << "Number entered is negative\n";
    }
    else {
        cout << "Number entered is non negative\n";
    }
    return 0;
}
```



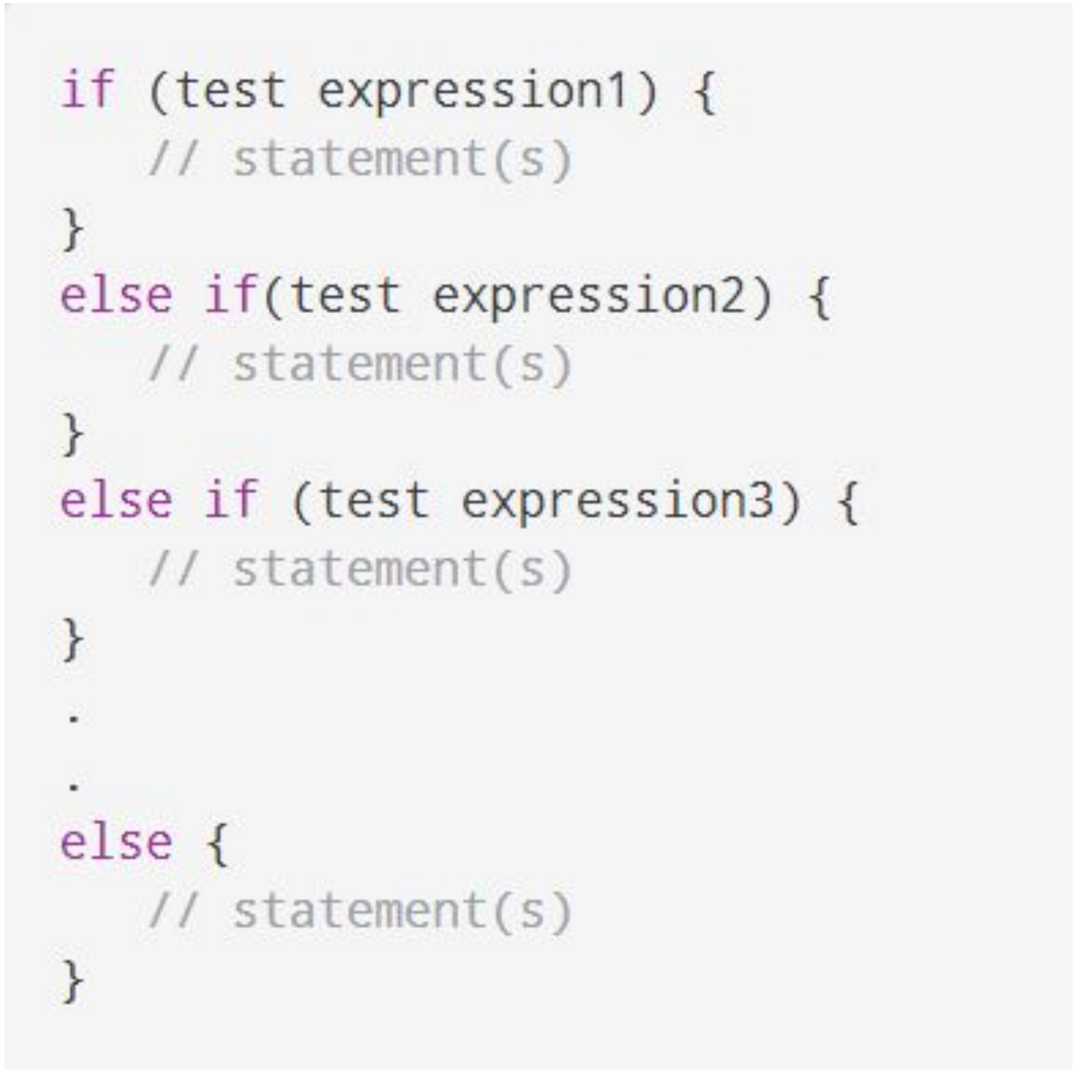
```cpp
#include<iostream>
using namespace std;
int main()
{
    int num;
    cout << "Enter a number: ";
    cin >> num;
    if(num<0) {
        cout << "Number entered is negative\n";
    }
    return 0;
}
```

IF ELSE PROGRAMS

IF ELSE LADDER

When a choice has to be made from more than 2 possibilities, the `if...else` ladder is used. The `if...else` statement executes a block of code depending upon whether the test expression is true or false.

The if...else ladder allows you to check between multiple test expressions and execute different statements.



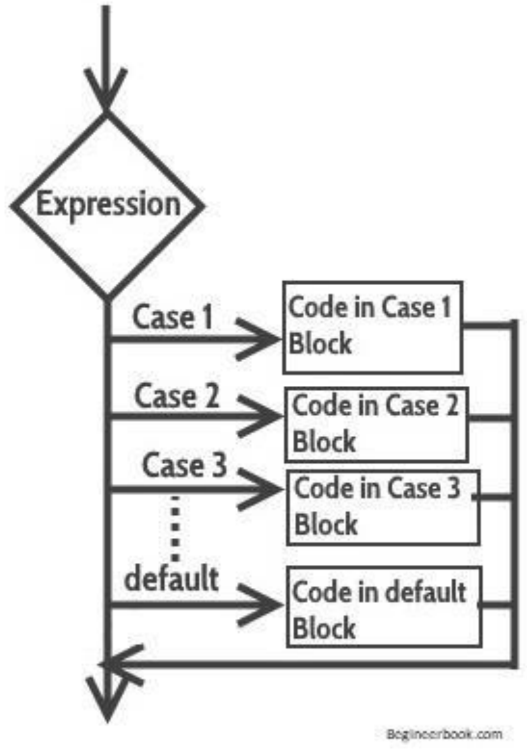
```
if (test expression1) {
   // statement(s)
}
else if(test expression2) {
   // statement(s)
}
else if (test expression3) {
   // statement(s)
}
.
.
else {
   // statement(s)
}
```

SWITCH CASE

The switch statement allows us to execute one code block among many alternatives.
You can do the same thing with the `if...else..if` ladder. However, the syntax of the `switch` statement is much easier to read and write.

Syntax :

```
switch (expression)
{
    case constant1:
      // statements
      break;

    case constant2:
      // statements
      break;
    .
    .
    default:
      // default
statements
}
```



Note:
The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.

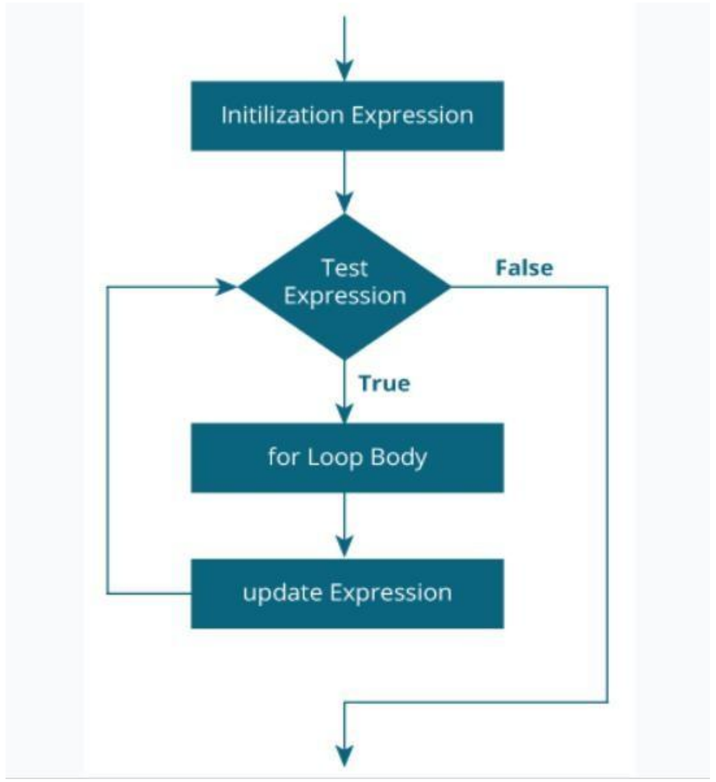
LOOPS IN C++

In programming, a loop is used to repeat a block of code until the specified condition is met.

C++ programming has three types of loops:

- for loop
- while loop
- do...while loop

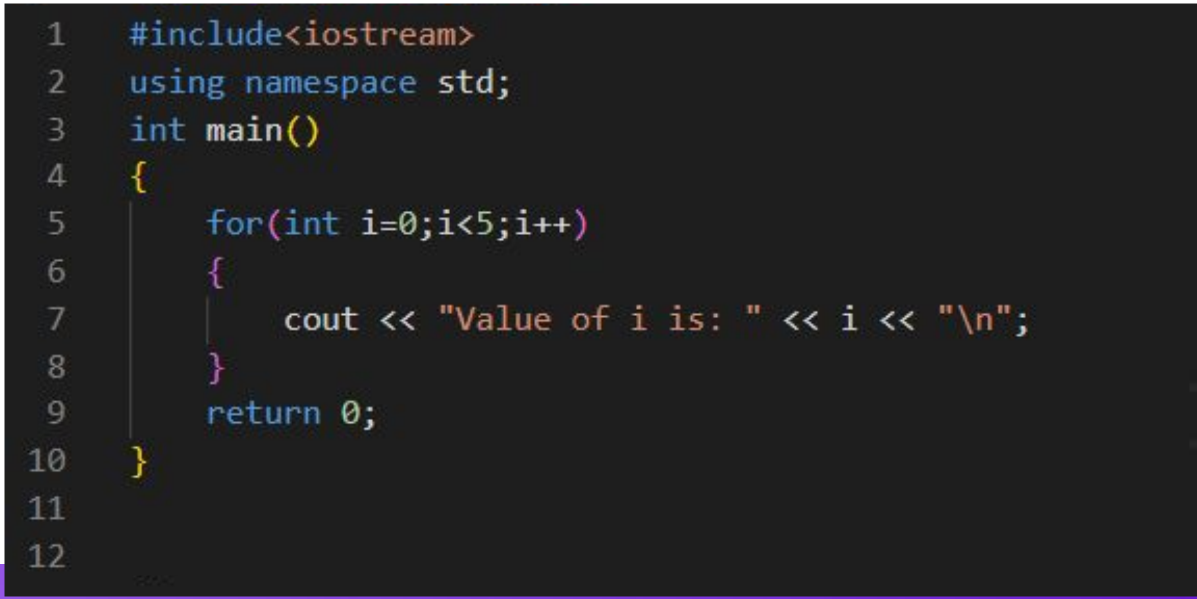
FOR LOOP



How for loop works?

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again, the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.

```
for (initializationStatement; testExpression; updateStatement)
{
    // statements inside the body of loop
}
```

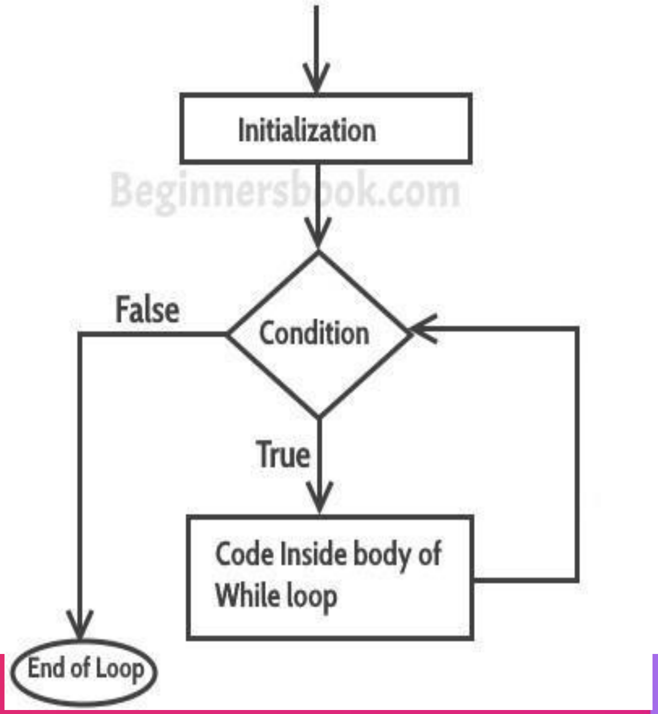


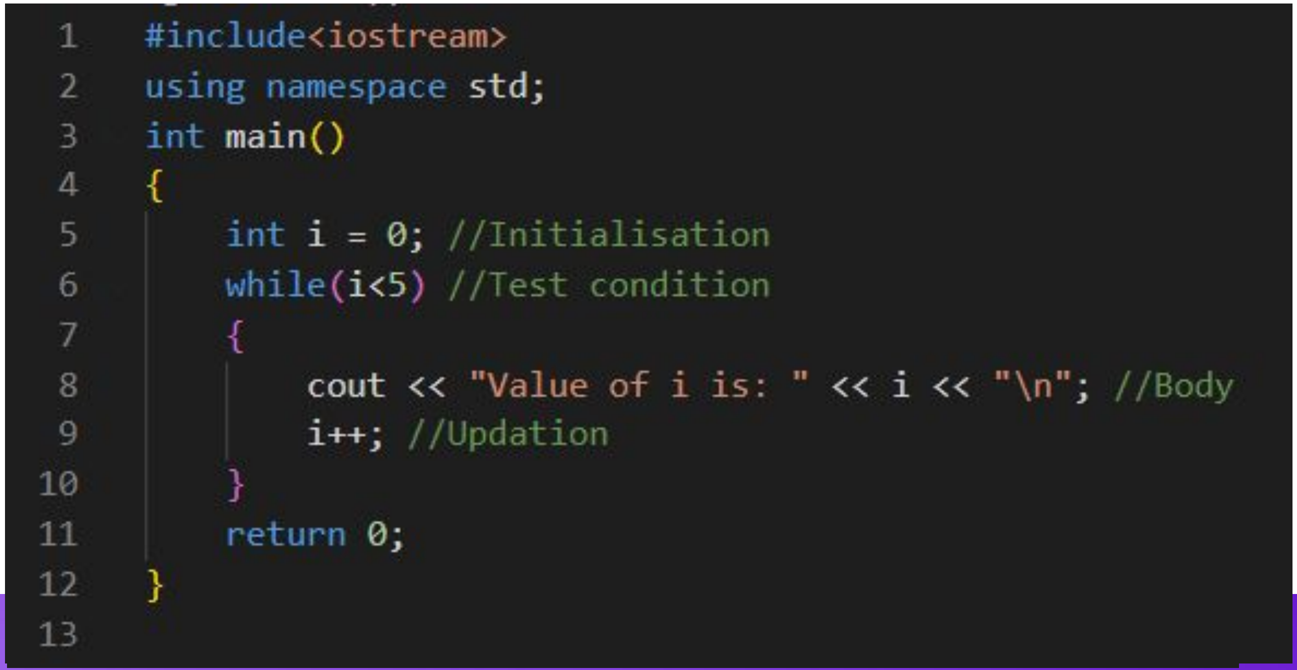
```
#include<iostream>
using namespace std;
int main()
{
    for(int i=0;i<5;i++)
    {
        cout << "Value of i is: " << i << "\n";
    }
    return 0;
}
```

WHILE LOOP

How while loop works?

- The initialization statement is kept out and executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the wile loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of while loop are executed. These also contains update statements if any.
- Again, the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.





```
#include<iostream>
using namespace std;
int main()
{
    int i = 0; //Initialisation
    while(i<5) //Test condition
    {
        cout << "Value of i is: " << i << "\n"; //Body
        i++; //Updation
    }
    return 0;
}

```

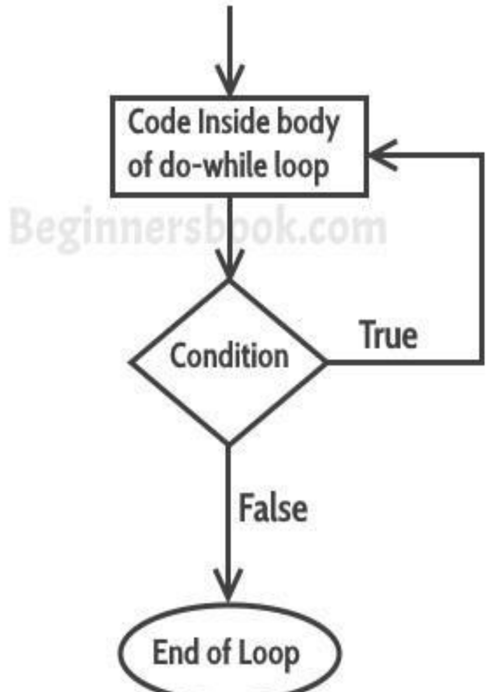
DO - WHILE LOOP

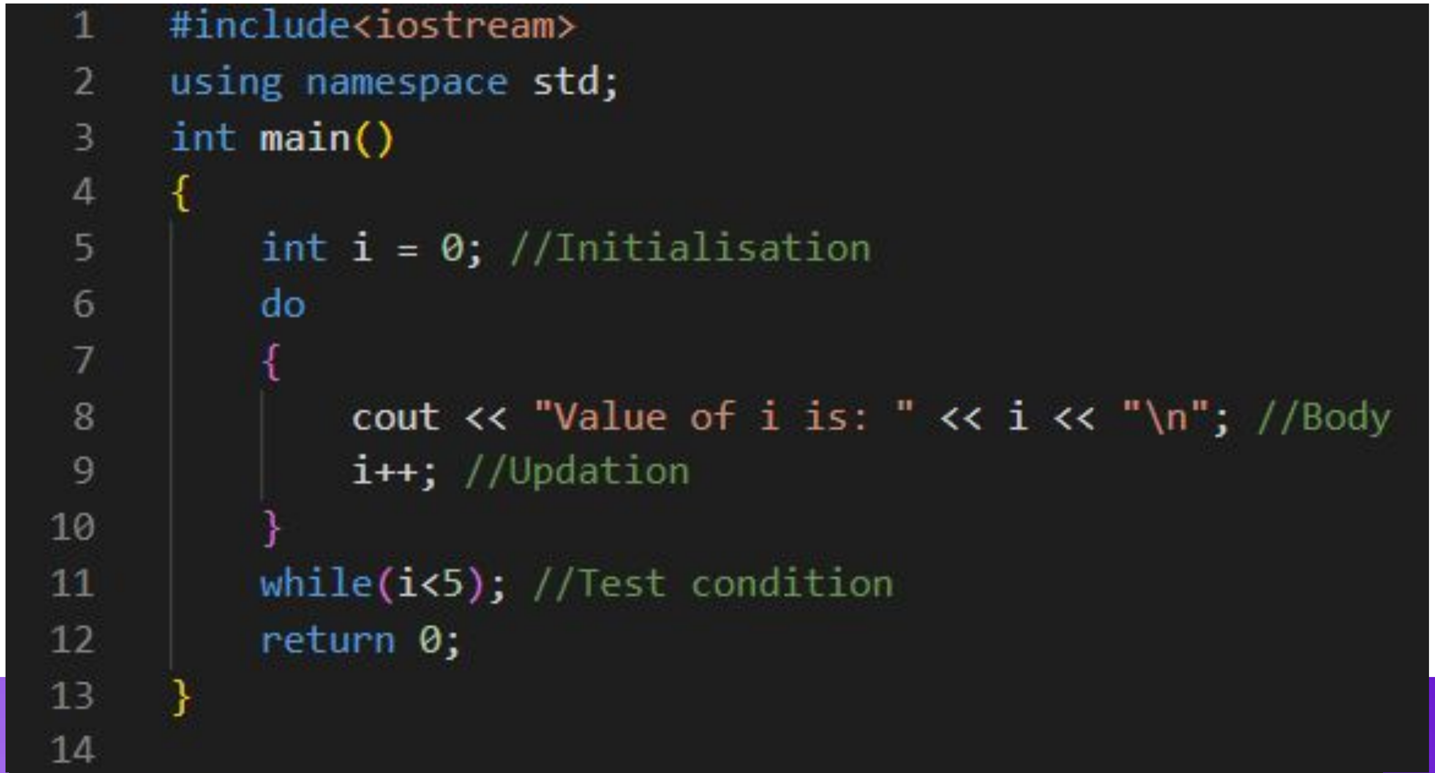
How DO while loop works?

- The initialization statement is kept out and executed only once.
- Then, the body of the loop gets executed first including any updation specified inside body
- Then, the body exits, and test expression specified in while() is executed

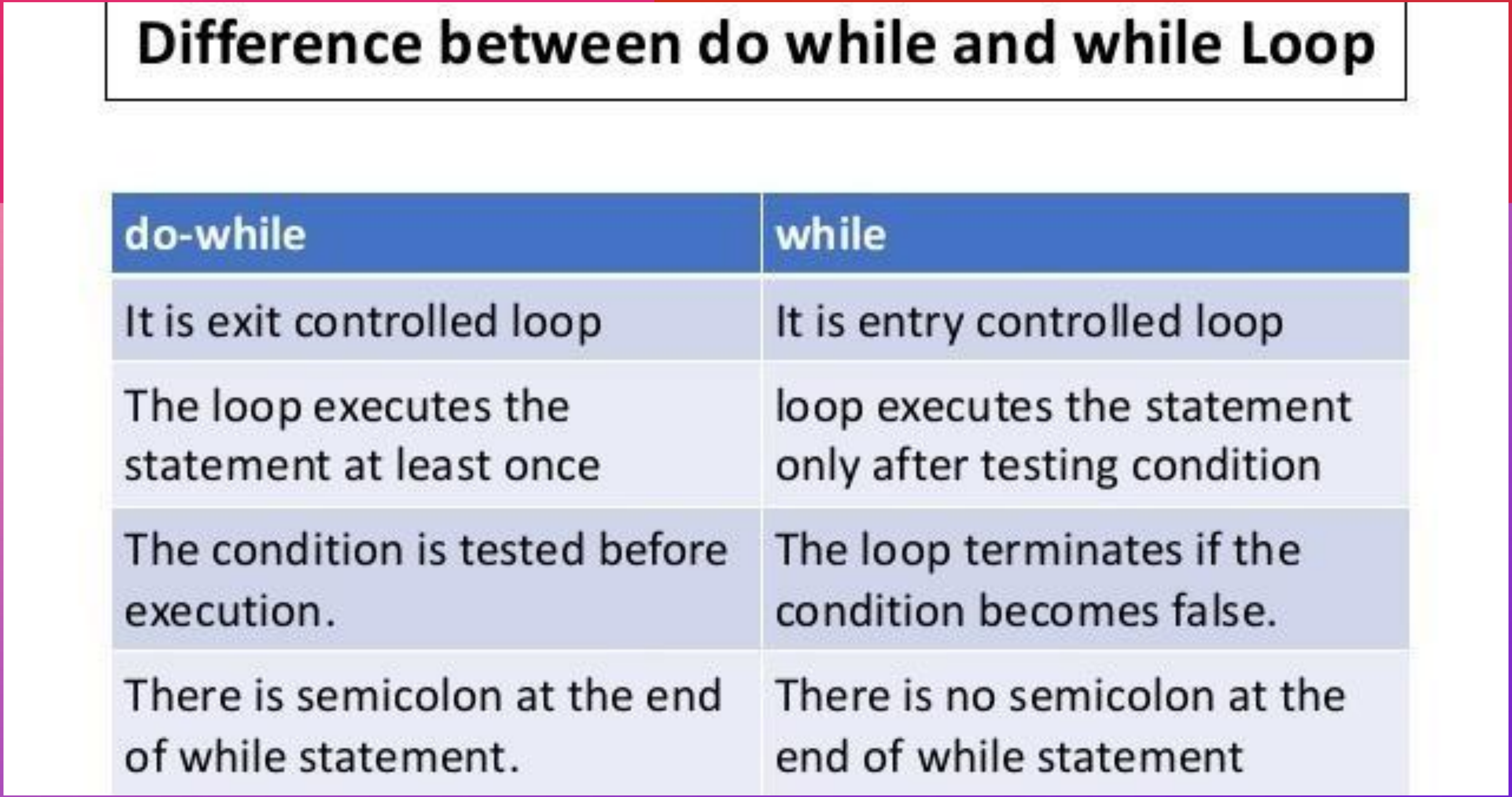
•If the test expression is true, again the statements inside body are executed and testing is done untill condition becomes false

- If the test expression is false, then the statements after while(); are executed.





```
#include<iostream>
using namespace std;
int main()
{
    int i = 0; //Initialisation
    do
    {
        cout << "Value of i is: " << i << "\n"; //Body
        i++; //Updation
    }
    while(i<5); //Test condition
    return 0;
}
```

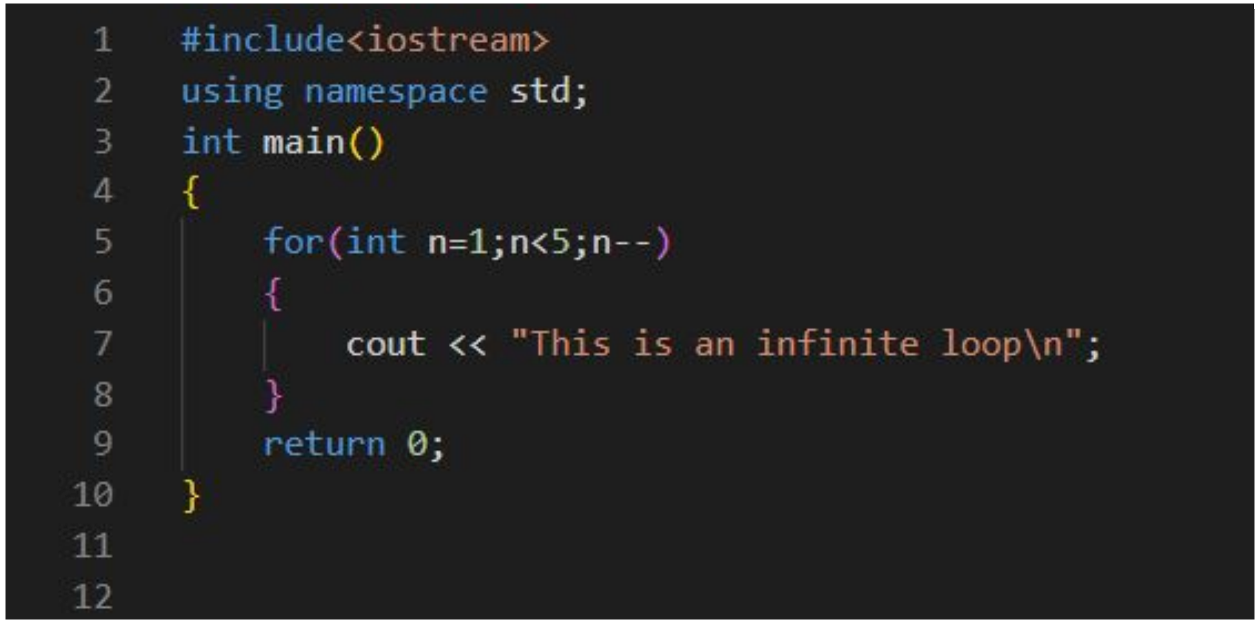


Difference between do while and while Loop

do-while	while
It is exit controlled loop	It is entry controlled loop
The loop executes the statement at least once	loop executes the statement only after testing condition
The condition is tested before execution.	The loop terminates if the condition becomes false.
There is semicolon at the end of while statement.	There is no semicolon at the end of while statement

INFINITE LOOP

The loops that go on executing infinitely and never terminate are called infinite loops. Sometimes we write these loops by mistake while sometimes we deliberately make use of these loops in our programs.



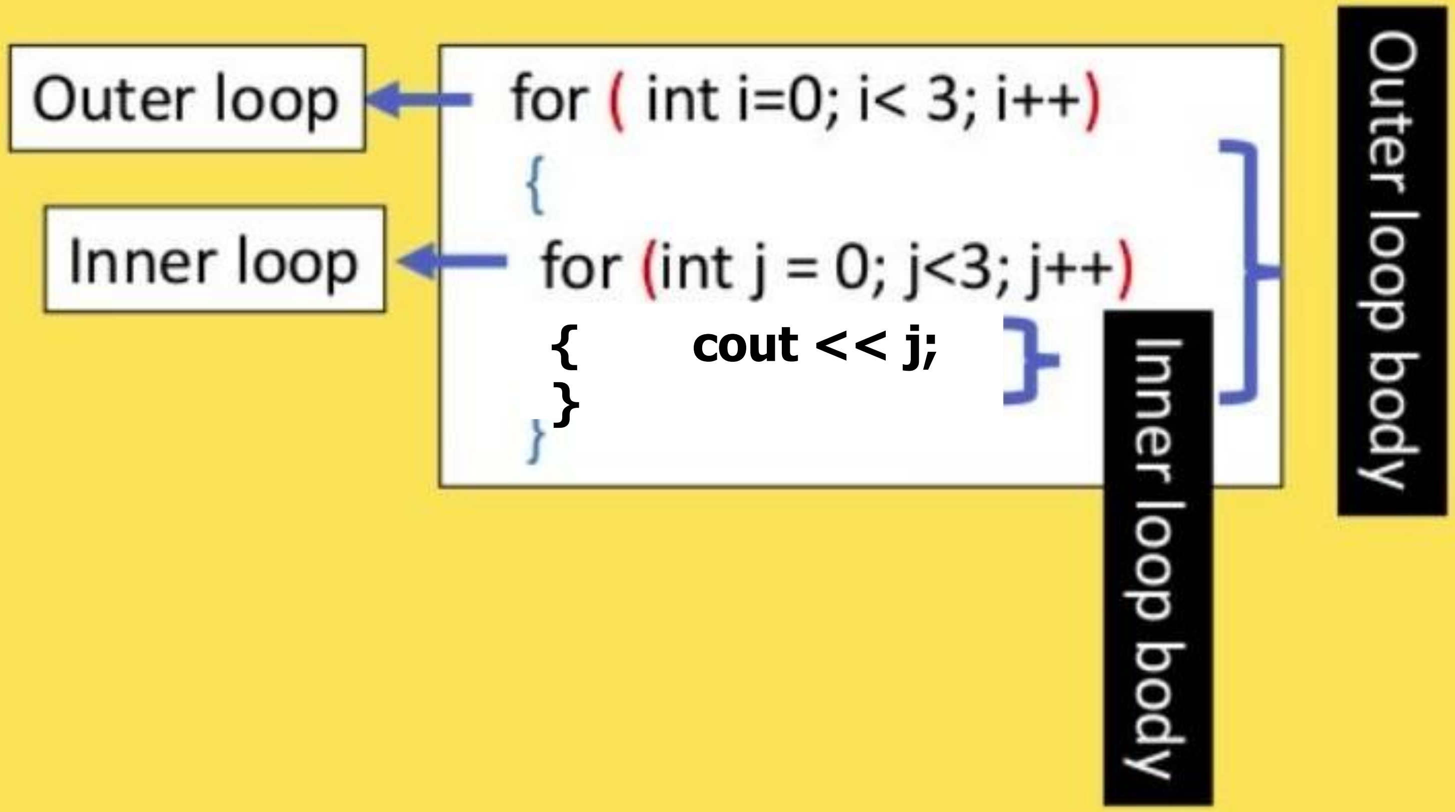
```
#include<iostream>
using namespace std;
int main()
{
    for(int n=1;n<5;n--)
    {
        cout << "This is an infinite loop\n";
    }
    return 0;
}


```

To stop the running of Infinite loop use: **ctrl+C**

It is important to stop the running of Infinite loop ASAP otherwise your system might get hanged.

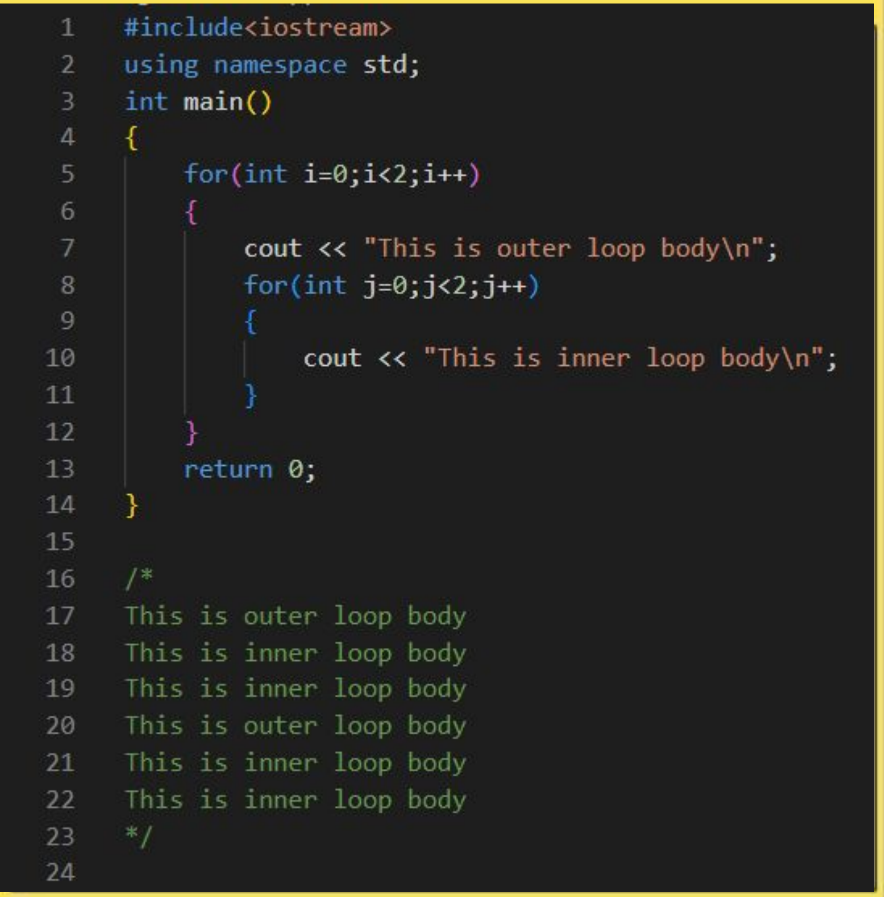
What are nested loops?

- A nested loop is a loop inside the body of another loop.
- The nested loop is known as the **inner loop** and the loop in which it is nested is known as the **outer loop**



```
for ( int i=0; i< 3; i++)
{
 for (int j = 0; j<3; j++)
 {      cout << j;
 }
}
```

Working of Nested Loops



```
#include<iostream>
using namespace std;
int main()
{
    for(int i=0;i<2;i++)
    {
        cout << "This is outer loop body\n";
        for(int j=0;j<2;j++)
        {
            cout << "This is inner loop body\n";
        }
    }
    return 0;
}

/*
This is outer loop body
This is inner loop body
This is inner loop body
This is outer loop body
This is inner loop body
This is inner loop body
*/
```

1. STEP 1:The outer loop is initialized with value of i as 0
2. STEP 2: Value of i is tested, since the condition is true(i<2), the loop is entered
3. STEP 3:A newline is displayed This is outer loop body This is part of outer loop.
4. STEP 4 :The control goes to inner loop, where j is initialized with 0
5. STEP 5: Value of j is tested, j<2 is true, inner loop is entered
6. STEP 6: The statement This is innerloop body is executed, value of j is displayed
7. STEP 7: The value of j is incremented.
8. Now STEPs 5, 6 and 7 are repeated till the condition ,j<2, becomes false.
9. When value of j is 2, control comes out of inner loop.
8. STEP 8: Now the control goes to outer loop update statement, i is incremented.
9. STEPs 2 – 7 are repeated. The steps are repeated for value of i = 1 This continues till value of i becomes 2. Then the outer loop is terminated.

PYRAMIDS

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

(a)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

(b)

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

(c)

```
        1
      1   1
    1   1   1
  1   1   1   1
1   1   1   1   1
```

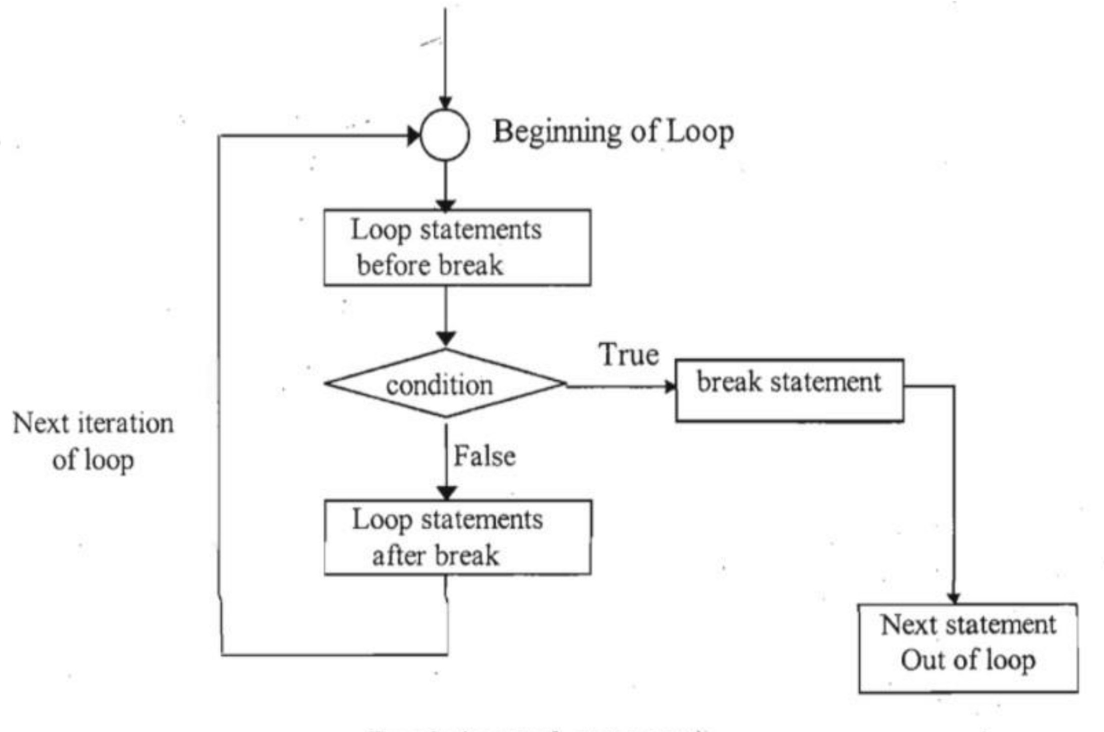
(d)

SOLUTION (B)

```
#include<iostream>
using namespace std;
int main() {
      for(int  i=1; i<=5; i++)  {
             for(int j=1; j<=i; j++)  {
                  cout << j << " ";
            }
            cout << "\n";
      }
      return 0;
}
```

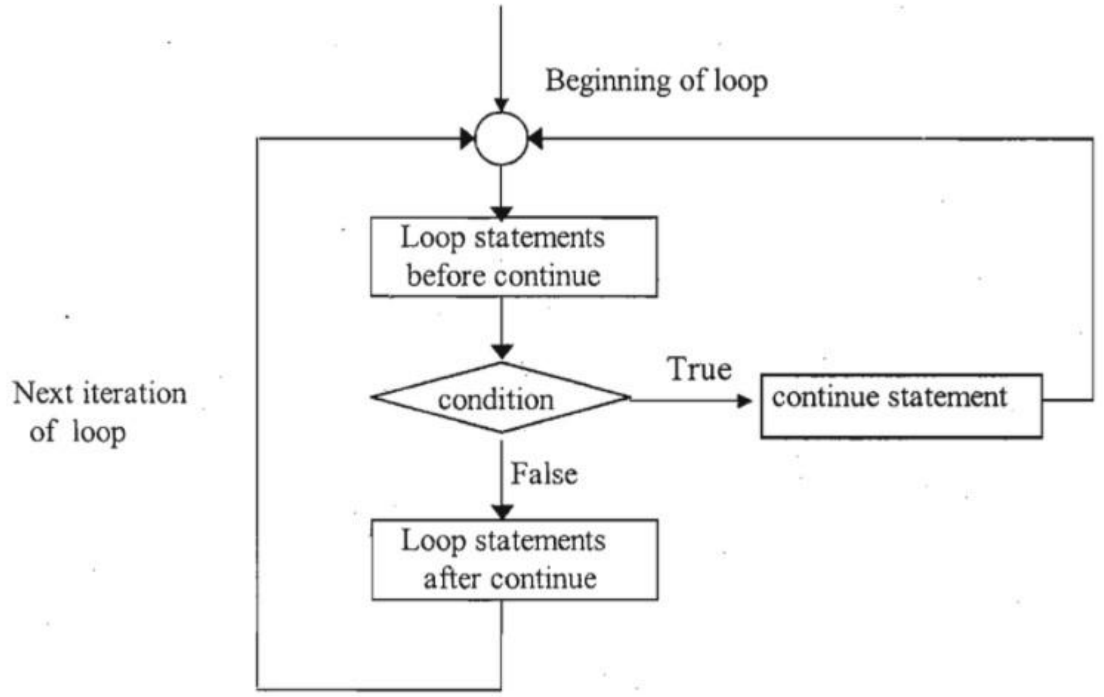
BREAK AND CONTINUE

Break: It is used to terminate the loop. This statement causes an immediate exit from that loop in which this statement appears, and the control is transferred to the statement immediately. after the loop.


Beginning of Loop
Loop statements before break
condition
True
break statement
False
Loop statements after break
Next iteration of loop
Next statement Out of loop

Break (control statement)

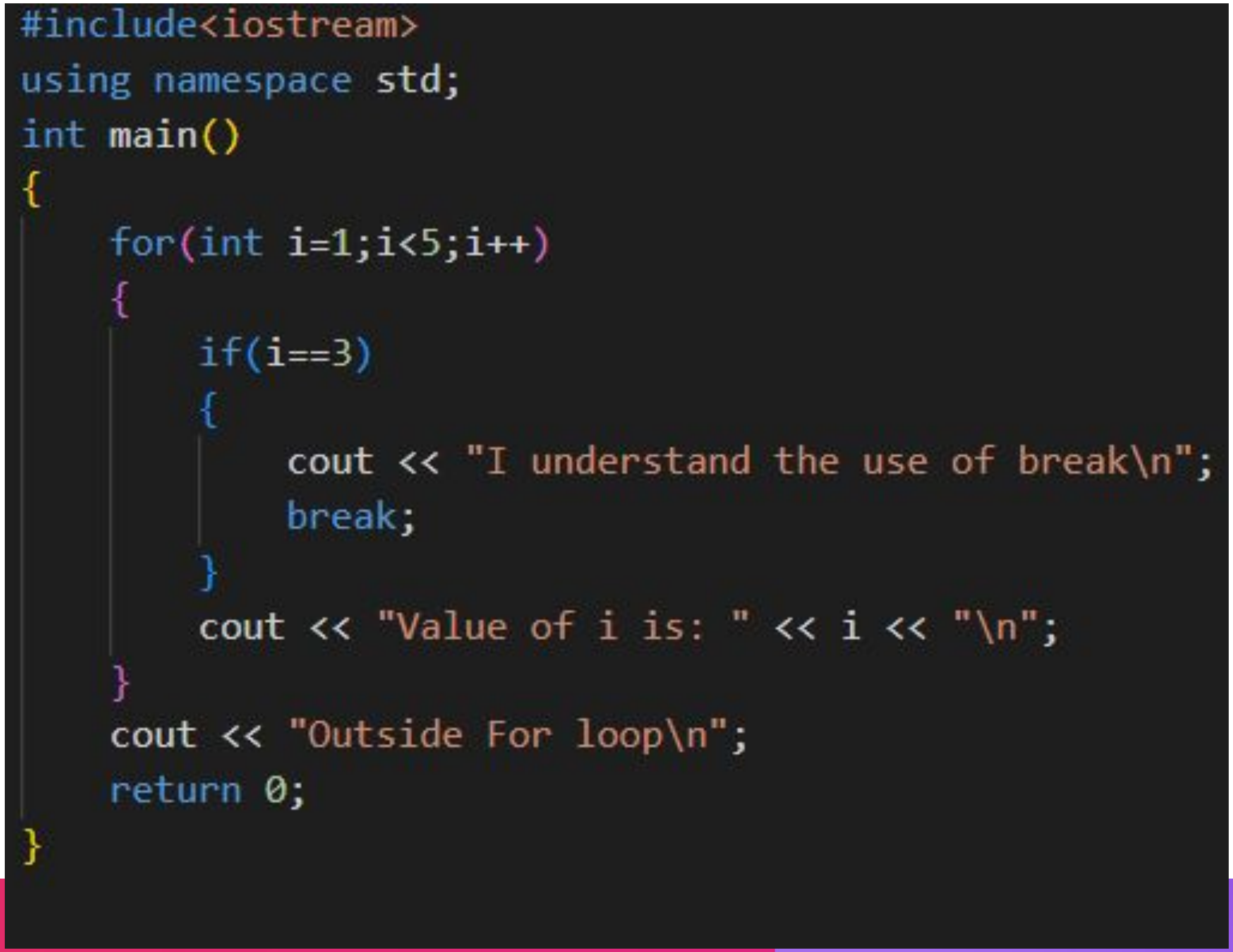
Continue: It is used when we want to go to the next iteration of the loop after skipping some statements of the loop.



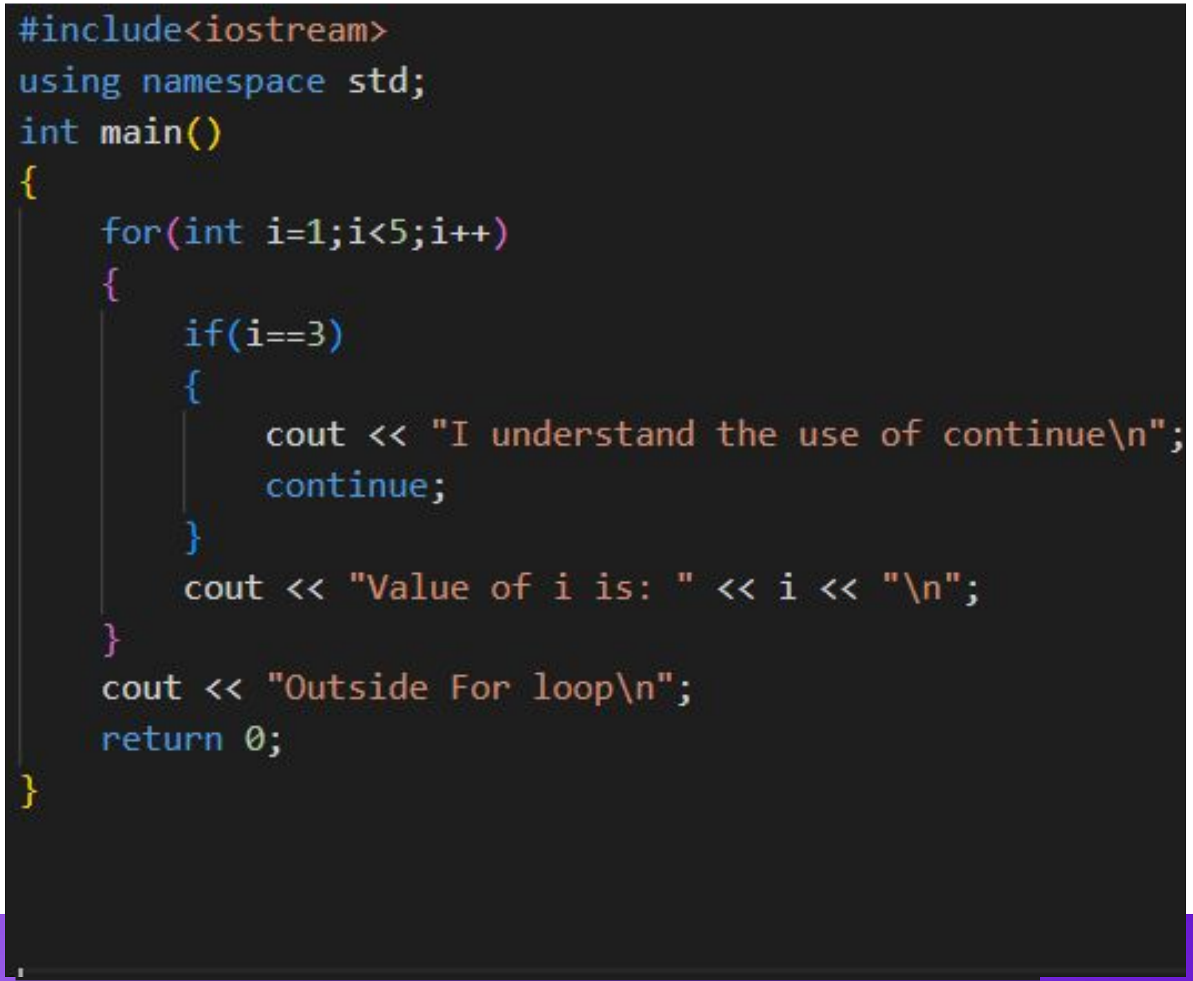
continue (control statement)

NOTE:

- In while and do-while loops, after continue statement the control is transferred to the test condition and then the loop continues
- Whereas in **for** loop after continue statement the control is transferred to update expression and then the condition is tested.



```cpp
#include<iostream>
using namespace std;
int main()
{
    for(int i=1;i<5;i++)
    {
        if(i==3)
        {
            cout << "I understand the use of break\n";
            break;
        }
        cout << "Value of i is: " << i << "\n";
    }
    cout << "Outside For loop\n";
    return 0;
}
```



```cpp
#include<iostream>
using namespace std;
int main()
{
    for(int i=1;i<5;i++)
    {
        if(i==3)
        {
            cout << "I understand the use of continue\n";
            continue;
        }
        cout << "Value of i is: " << i << "\n";
    }
    cout << "Outside For loop\n";
    return 0;
}
```