do while Loop

If the conditional expression controlling a **while** loop is initially false, then the body of the loop will not be executed at all. However, sometimes it is desirable to execute the body of a **while** loop at least once, even if the conditional expression is false to begin with. The **do-while** loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is:

Syntax

```
Initialization
do
{
        // body of loop
        Increment/ decrement
} while (condition);
```

Example 1: What is the program supposed to output?

```
#include <iostream>
using namespace std;
int main() {
int i = 1;
do{
cout<<i<<"\n";
i++;
} while (i <= 10) ;
}
```

Exercise 1: What changes will you need to make to the above program to get output 1, 4, 7, 10?

Exercise 2: What do you think will happen if you put i++ after the while (i <= 10) ; ?

Take Away Exercise

1. Draw a flowchart that shows execution of do while loop
2. Using the do while loop write a C++ program to find the sum of positive numbers

FOR loop

Syntax

```
for(initialization; condition; iteration)
{
        // body
}
```

The **for** loop operates as follows:

When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop. It is important to understand that the initialization expression is only executed once.

Next, condition is evaluated. This must be a Boolean expression. It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.

Next, the iteration portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable. The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

Example: What output will this program display?

```
#include <iostream>
using namespace std;
int main(){
 for(int i=1; i<=6; ++i){
     cout<<"Hello there"<<endl;

  }
  return 0;
}
```

Exercise 1: What output will this program display?

```
#include <iostream>
using namespace std;
int main(){
   for(int i=1; i<=6; i++){
      cout<<i<<endl;
   }
   return 0;
}
```

Exercise 2: Modify the program above to display 6, 5, 4, 3, 2, 1 in that order

Take away exercise

Draw a flowchart to show execution of a FOR loop

Using FOR loop write a C++ program to find the sum of first n natural numbers

Using FOR loop write a C++ program to find n factorial

Nested FOR loop

A nested FOR loop is a loop inside another loop statement. The number of loops depend on the complexity of a problem. Suppose, a loop, outer loop, running *n* number of times consists of another loop inside it, inner loop, running *m* number of times. Then, for each execution of the outer loop from 1…n, the inner loop runs maximum of m times.

Syntax for nested FOR loop

```
for ( initial; condition; increment ) {
   for ( initial; condition; increment ) {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

Example 1. Predict the output of the program?

```
#include <iostream>
using namespace std;
int main() {
  int rows = 5;
  int columns = 3;
  for (int i = 1; i <= rows; ++i) {
    for (int j = 1; j <= columns; ++j) {
      cout << "*  ";
    }
    cout << endl;
  }
  return 0;
}
```

Example 2: What will be the output?

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

Class Exercise

1. Modify the program in example 2 to display the following patterns

   1

   1 2

   1 2 3

   1 2 3 4

   1 2 3 4 5

2. Modify the program in example 2 to display the following patterns

   1

   2 2

   3 3 3

   4 4 4 4

   5 5 5 5 5

Take Away Exercise

Using FOR loop write a C++ program to display the following patterns

```
1   2   3   4   5
2   3   4   5   6
3   4   5   6   7
4   5   6   7   8
5   6   7   8   9
```

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

Jump Statements

Jump statements allows program to execute in a nonlinear fashion. They are used to interrupt the normal flow of program. Jump statements cause an unconditional jump to another statement elsewhere in the code. They are used primarily to interrupt switch statements and loops. C++ offers break, continue and goto jump statements

Break statement: A break statement takes control out of the loop. The break statement is used inside a loop or switch/case statement. When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Example : Predict the output of the program

```
#include<iostream>
using namespace std;
    main()
    {
        int a=1;
        while(a<=10)
        {
            if(a==5)
              break;
             cout << "\nStatement " << a;
            a++;
         }
             cout << "\nEnd of Program.";
    }
```

Continue statement: A continue statement takes control to the beginning of the loop or simply put, to the next iteration.

Example: Predict the output of the program

```
#include<iostream.h>
    void main()
    {
        int a=0;
        while(a<10)
         {
           a++;
            if(a==5)
              continue;
             cout << "\nStatement " << a;

         }
            cout << "\nEnd of Program.";
    }
```

Goto statement: A goto Statement take control to a desired line of a program. It jumps from one point to another point within a function. It allows the program's execution flow to jump to a specified location within the function. There are two ways to call the goto statement.

Syntax 1	Syntax 2
goto label;	label: ;
//block of statements	//block of statements
label: ;	goto label;

Example: (goto statement based on the first syntax) Predict the output

```
//based on syntax 1
#include<iostream>
using namespace std;
int main()
{
int i, j;
i=2;j=5;
if(i>j)
goto iGreater;
else
goto jGreater;
iGreater:
cout<<i<<" is greater";
jGreater:
cout<<j<<" is greater";
return 0;
}
```

Example: (goto statement based on the second syntax) Predict the output

```
//based on Syntax 2
#include <iostream>
using namespace std;
int main()
{
int n = 1;
print:
cout << n << " ";
n++;
if (n <= 5)
goto print;
return 0;
}
```

Disadvantages of Goto statements

Early programming languages like FORTRAN and early versions of BASIC did not have structured statements like while, so programmers were forced to use goto statements to write loops. The problem with using goto statements is that **it is easy to develop program logic that is very difficult to understand**, even for the original author of the code.

It is easy to get caught in an **infinite loop.**