LAB 08: Control Structures (for/while/do-while loop)

Objective(s): Upon completion of this lab session, learners will be able to:
CLOs: CLO1, CLO4

The increment and decrement operators
++ and -- are operators that add and subtract 1 from their operands. To *increment* a value means to increase it by one, and to *decrement* a value means to decrease it by one. Both of the following statements increment the variable num:

num = num + 1;
num += 1;
And num is decremented in both of the following statements:
num = num - 1;
num = 1;

C++ provides a set of simple unary operators designed just for incrementing and decrementing variables. The increment operator is ++ and the decrement operator is --. The following statement uses the ++ operator to increment num: *num++;* And the following statement decrements num: *num--;*

Example:

```
// This program demonstrates the ++ and -- operators.
 #include <iostream>
 using namespace std;
 int main()
 {       int num=4;
         // Display the value in num.
         cout << "The variable num is " << num << endl;
         cout << "I will now increment num.\n\n";
         // Use postfix ++ to increment num.
         num++;
         cout << "Now the variable num is " << num << endl;
         cout << "I will increment num again.\n\n";
         // Use prefix ++ to increment num.
         ++num;
         cout << "Now the variable num is " << num << endl;
         cout << "I will now decrement num.\n\n";

         // Use postfix -- to decrement num.
         num--;
         cout << "Now the variable num is " << num << endl;
         cout << "I will decrement num again.\n\n";
}
```

Program Output:

```
The variable num is 4
I will now increment num.
Now the variable num is 5
I will increment num again.
Now the variable num is 6
I will now decrement num.
Now the variable num is 5
I will decrement num again.
Now the variable num is 4
```

The Difference between Postfix and Prefix Modes:

It doesn't matter if the increment or decrement operator is used in postfix or prefix mode. The difference is important, however, when these operators are used in statements that do more than just incrementing or decrementing. For example, look at the following lines:

```
num = 4;
cout << num++;
```

This cout statement is doing two things: (1) displaying the value of num, and (2) incrementing num. But which happens first? cout will display a different value if num is incremented first than if num is incremented last. The answer depends on the mode of the increment operator. Postfix mode causes the increment to happen after the value of the variable is used in the expression. In the example, cout will display 4, then num will be incremented to 5. Prefix mode, however, causes the increment to happen first. In the following statements, num will be incremented to 5, then cout will display 5:

```
num = 4;
cout << ++num;
```

Loops:

A loop is part of a program that repeats. A *loop* is a control structure that causes a statement or group of statements to repeat. C++ has three looping control structures: the while loop, the do-while loop, and for loop. The difference between these structures is how they control the repetition.

The while Loop:

The while loop has two important parts: (1) an expression that is tested for a true or false value, and (2) a statement or blocks that is repeated as long as the expression is true.

Here is the general format of the while loop:

Syntax:

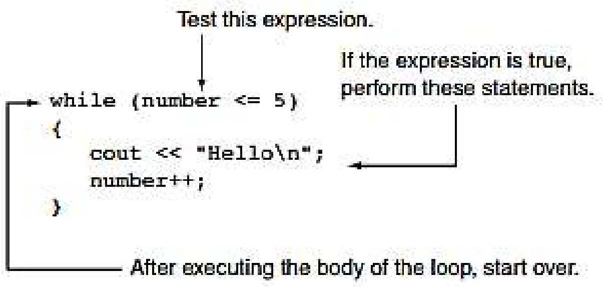
```
while (expression)
{
statement;
statement;
// Place as many statements here
// as necessary.
}
```

Example: // This program demonstrates a simple while loop.

```
#include <iostream>
using namespace std;
int main()
{
        int number = 1;
        while (number <= 5) {
        cout << "Hello\n";
        number++;
        }
        cout << "That's all!\n";
        return 0;
}
```

Program Output:

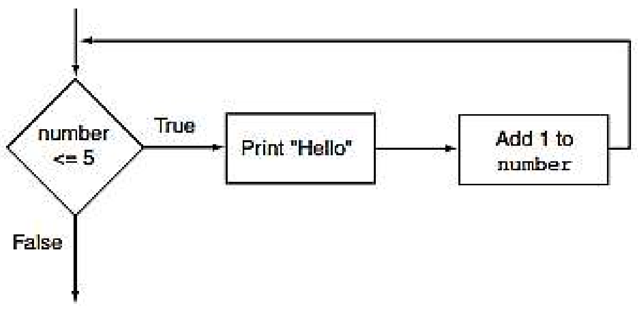
```
Hello
Hello
Hello
Hello
Hello
That's all!
```



```
while (number <= 5)
{
   cout << "Hello\n";
   number++;
}
```

Test this expression.
If the expression is true, perform these statements.
After executing the body of the loop, start over.

Flow diagram:



In this example, the number variable is referred to as the *loop control variable* because it controls the number of times that the loop iterates.

Counters:

A counter is a variable that is regularly incremented or decremented each time a loop iterates. Sometimes it’s important for a program to control or keep track of the number of iterations a loop performs. For example, Program in example displays a table consisting of the numbers 1 through 10 and their squares, so its loop must iterate 10 times.

Example:

```
// This program displays the numbers 1 through 10 and
 // their squares.
 #include <iostream>
 using namespace std;
 int main()
{

   int num =1; // initializing the counter
   cout << "Number  Squared\n";

   cout << "-------------------------\n";
  while (num <= 10){
            cout << num << "\t\t" << (num * num) << endl;
            num++; //Increment the counter.
            num++; //Increment the counter.
            }
   return 0;
 }
```

Program Output:

```
Program Output
Number Squared
-------------------------
1          1
2          4
3          9
4          16
5          25
6          36
7          49
8          64
```

9 81

The do-while Loop

The do-while loop is a posttest loop, which means its expression, is tested after each iteration.

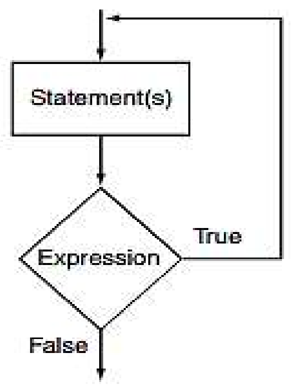
Syntax:

```
do
statement;
while (expression);
```

Here is the format of the do-while loop when the body of the loop contains multiple statements:

```
do
{
statement;
statement;
// Place as many statements here
// as necessary.
} while (expression);
```

Flow diagram:



Example:

```
// This program averages 3 test scores. It repeats as
 // many times as the user wishes.
 #include <iostream>
 using namespace std;

 int main()
 {
            int score1, score2, score3; // Three scores
            double average; // Average score

           char again; // To hold Y or N input

       do{
                   // get three scores

           Cout<<"enter 3 scores and I will avg them";

           Cin>>score1>>socre 2>>score3;
           /calculate and display the avg.

           average = (score1 + score2 + score3) / 3.0;
           cout << "The average is " << average << ".\n";

           // Does the user want to average another set?

           cin >> again;
         }

 while (again == 'Y' || again == 'y');

  return 0;
 }
```

Program Output:

```
Enter 3 scores and I will average them: 80 90 70 [Enter]
The average is 80.
Do you want to average another set? (Y/N) y [Enter]
Enter 3 scores and I will average them: 60 75 88 [Enter]
The average is 74.3333.
Do you want to average another set? (Y/N) n [Enter]
```

The for Loop:
The for loop is ideal for performing a known number of iterations. A count-controlled loop must possess three elements:

It must initialize a counter variable to a starting value.

It must test the counter variable by comparing it to a maximum value. When the counter variable reaches its maximum value, the loop terminates.

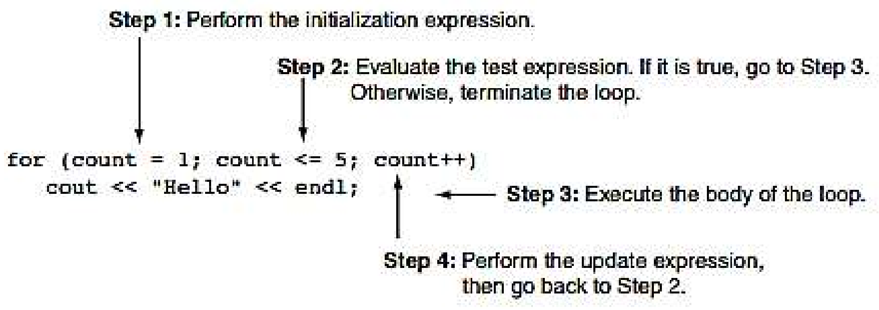
It must update the counter variable during each iteration. This is usually done by incrementing the variable.

Here is the format of the for loop when it is used to repeat
a single statement:

```
for (initialization; test;
update)
```

The format of the for loop when it is used to repeat a block is

```
for (initialization; test; update)
{
statement;
statement;
// Place as many statements here
// as necessary.
}
```



Step 1: Perform the initialization expression.

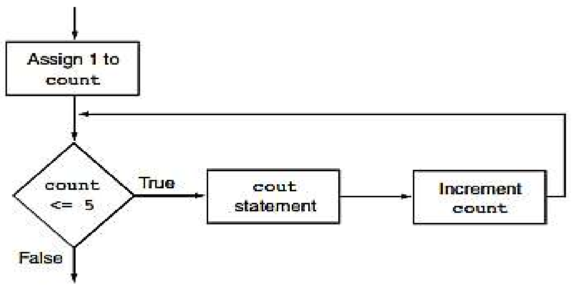
Step 2: Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.

```
for (count = 1; count <= 5; count++)
   cout << "Hello" << endl;
```

Step 3: Execute the body of the loop.

Step 4: Perform the update expression, then go back to Step 2.

Flow Diagram:



Example: // This program displays the numbers 1 through 10 and // their squares.

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

 int main()
 {
	int num;
	cout << "Number  Squared\n";
	cout << "-------------------------\n";
	for (num = 1; num <= 10; num++)
		cout << num << "\t\t" << (num * num) << endl;
	return 0;
}
```

Nested Loop:

A loop that is inside another loop is called a *nested loop.*

Example:

```
// This program averages test scores. It asks the user for the
// number of students and the number of test scores per student.
 #include <iostream>
 #include <iomanip>
 using namespace std;

Int main(){

	int numStudents, // Number of students
	numTests; // Number of tests per student

	double total, // Accumulator for total scores
	double total, // Accumulator for total scores
}




        average; // Average test score
        // Set up numeric output formatting.

        cout << fixed << showpoint << setprecision(1);
        // Get the number of students.

        cout << "This program averages test scores.\n";

        cout << "For how many students do you have scores? ";
        cin >> numStudents;

        // Get the number of test scores per student.
        cout << "How many test scores does each student have? ";

        cin >> numTests;

        // Determine each student's average score.
        for (int student = 1; student <= numStudents; student++)

        {
                total = 0; // Initialize the accumulator.

                for (int test = 1; test <= numTests; test++)

                {
                        double score;

                        cout << "Enter score " << test << " for ";
                }

                 cout << "student " << student << ": ";
                cin>>score;

                total += score;

        }
        average = total / numTests;

        cout << "The average score for student " << student;

        cout << " is " << average << ".\n\n";


        return 0;
}
```

Program Output:

```
This program averages test scores.

For how many students do you have scores? 2 [Enter]
How many test scores does each student have? 3 [Enter]
Enter score 1 for student 1: 84 [Enter]
Enter score 2 for student 1: 79 [Enter]
Enter score 3 for student 1: 97 [Enter]
The average score for student 1 is 86.7.
Enter score 1 for student 2: 92 [Enter]
Enter score 2 for student 2: 88 [Enter]
Enter score 3 for student 2: 94 [Enter]
The average score for student 2 is 91.3.
```

Breaking Out of a Loop

The break statement causes a loop to terminate early. Sometimes it's necessary to stop a loop before it goes through all its iterations. The break statement, which was used with switch, can also be placed inside a loop. When it is encountered, the loop stops and the program jump to the statement immediately following the loop. The while loop in the following program segment appears to execute 10 times, but the break statement causes it to stop after the fifth iteration,

Lab Tasks:

Task 1

Write a program that uses a loop to display the characters for the ASCII codes 0 to 127. Display 16 characters on each line.

Task 2

Write a program using while loop to find the sum of the following series. 1+1/2+1/3+…..1/45

Task 3

By applying for loop, write a C++ program that prints

A sequence of numbers in ascending order from 1 to 100 (inclusive).

Modify your program in part (a) to make it prints odd numbers within 1 to 100.

Write a C++ Program that receives a positive integer (N) and prints a series of numbers from 1 to N (inclusive) in ascending order.

Write a C++ Program that displays a series of alphabets in descending order from 'Z' to 'A'.

Modify your program in part (d) so that the program will display consonants only, no vowels.

Write a C++ program that receives start value and end value. Then, your program will display a series of numbers from the start value to the end value inclusive.

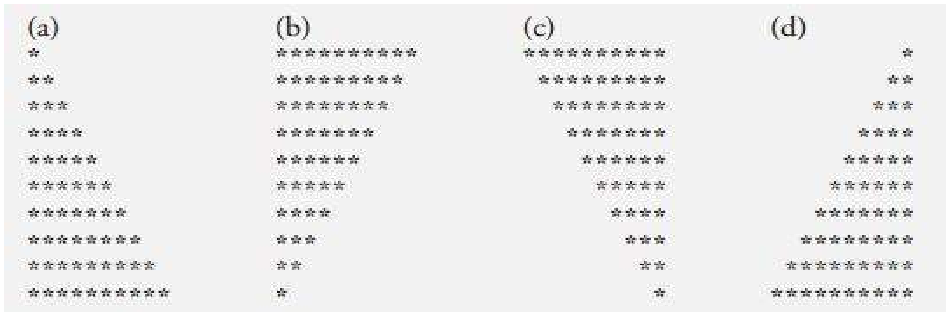
Task 4

Perform Task 5 from Lab 4 using Do While Loop.

Task 5

(Drawing Patterns with Nested for Loops)

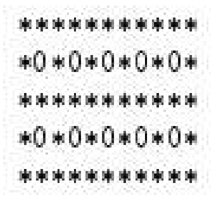
Write a program that uses for statements to print the following patterns separately, one below the other. Use for loops to generate the patterns. All asterisks (*) should be printed by a single statement of the form cout « '*'; (this causes the asterisks to print side by side). [Hint: The last two patterns require that each line begin with an appropriate number of blanks. Extra credit: Combine your code from the four separate problems into a single program that prints all four patterns side by side by making clever use of nested for loops.



```
(a)           (b)           (c)           (d)
*             **********    **********             *
**            *********      *********            **
***           ********        ********           ***
****          *******          *******          ****
*****         ******            ******         *****
******        *****              *****        ******
*******       ****                ****       *******
********      ***                  ***      ********
*********     **                    **     *********
**********    *                      *    **********
```

Task 6

Draw a pattern of 0’s surrounded by *’s. The pattern should be like.



```
***********
*0*0*0*0*0*
***********
*0*0*0*0*0*
***********
```