Lab # 06

LOOPS

6.1 Objective:

Learn about the primary mechanism for telling a computer that a sequence of tasks needs to be repeated a specific number of times.

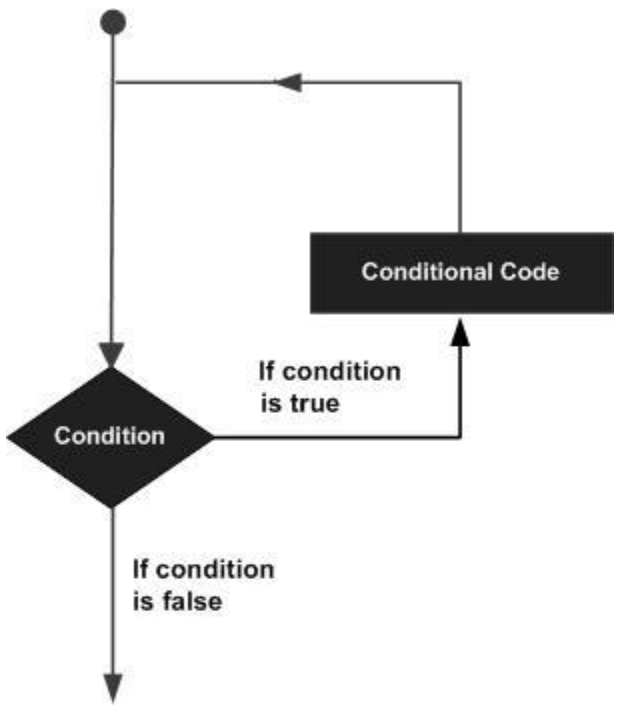
6.2 Scope:

The student should know the following:

1. Requirement to add a number to itself ten times.
2. To execute a statement or group of statements multiple times.

6.3 Useful Concepts:

Loop statements are the primary mechanism for telling a computer that a sequence of tasks needs to be repeated a specific number of times. Suppose, for example, that you have a requirement to add a number to itself ten times. A loop statement allows us to execute a statement or group of statements multiple times.



Why LOOPS are important?

Loops are relatively easy to implement, yet they can save you a lot of coding.

LOOPS can be of different types:

LOOPS can be divided as:

1. **While Loop:**

   Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

2. **Do while Loop:**

   Like a while statement, except that it tests the condition at the end of the loop body.

3. **For Loop:**

   Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

4. **Nested Loop:**

   You can use one or more loops inside any another while, for or do. While loop.

   Loops can further be distinguished as:

- Continue Statement: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.Function returning no value but accepting one or more arguments.
- Break Statement: Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.

For loop:

- The for() loop is basically three statements in one line. Within the parentheses following the "for", you have the initial expression, condition, and the loop expression. These are separated by semicolons.
- Syntax:
- for (''initializer''; ''conditional expression''; ''loop expression'')
- {
-     // statements to be executed
- }
- The initializer typically initializes a counter variable. Traditionally the variable name i is used for this purpose, though any valid variable name will do.
- Example:
- i = 0;
- The conditional expression specifies the test to perform to verify whether the loop has been performed the required number of iterations. For example, if we want to loop 10 times:

- i < 10;
- Finally the loop expression specifies the action to perform on the counter variable. For example to increment by 1:
- i++;
- The body of statements to be executed on each iteration of the loop is contained within the code block defined by the opening ({) and closing (}) braces. If only one statement is to be executed the braces are optional, though still recommended for code readability and so that you don't forget the add them if you later increase the number of statements to be performed in the loop.

While Loop:

The while() loop is better suited for cases where the program does not know how many times it is to be run, and instead checks for a different condition.

- Syntax:

```
while (input >= 0) { // "Greater-than or equal to zero" is the same as "not negative"
      sum = sum + input;
      printf("%d", &input);
   }
```

○

DO While Loop:

In a regular for() or while() loop, the condition is checked first, then the loop is executed. This means that if the initial condition was false to begin with, the body of the loop would never ever get executed; the code would skip directly over the loop. The do-while loop is guaranteed to perform one iteration of the loop first, before checking the condition. From that point on, as long as the condition remains true, the loop will continue to execute.

- Syntax:

```
 int n = 1;
do {
  sum = sum + n;
   n = n + 1;
} while (n <= 5)
```

Break and Continue Statement:

There are two statements (break; and continue ;) built in C programming to alter the normal flow of program. Loops are used to perform repetitive task until test expression is false but sometimes it is desirable to skip some statement/s inside loop or terminate the loop immediately with checking test condition. On these type of scenarios, continue; statement and break; statement is used respectively. The break statement is also used to terminate switch statement.

- **Syntax:**

break;

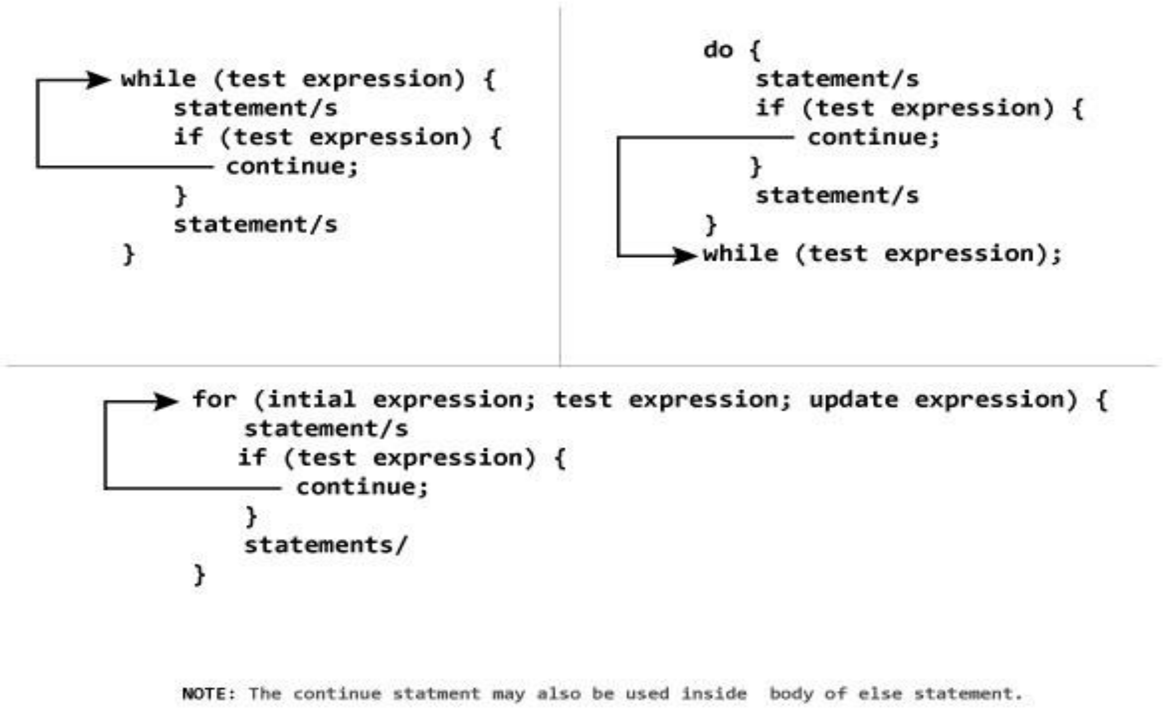
It is sometimes necessary to skip some statement/s inside the loop. In that case, continue; statement is used in C++ programming.

Syntax:

continue;

Working Body:

Syntax:



```
while (test expression) {
     statement/s
     if (test expression) {
          continue;
     }
     statement/s
}
```

```
do {
     statement/s
     if (test expression) {
          continue;
     }
     statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
     statement/s
     if (test expression) {
          continue;
     }
     statements/
}
```

NOTE: The continue statment may also be used inside body of else statement.

6.4 Examples

Example - 1:- Write a program Show all odd numbers using a for loop.

#include <iostream>

using namespace std;

```
int main()
{
        for (int count = 1; count <= 41; count += 2)
        {
                cout << count << ", ";
        }
        cout << endl;
        return 0;
}
```

Output:

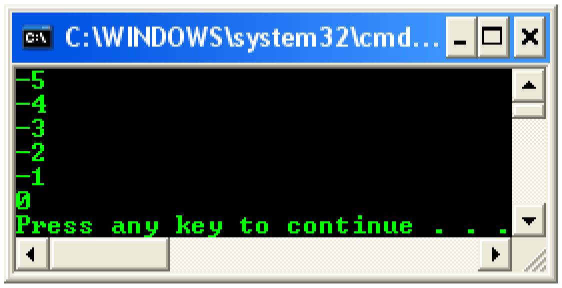
```
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,
```

Example - 2:- Write a program using do-while loop to display following output.

```
-5
-4
-3
-2
-1
0
```

```
# include <stdio.h>
# include<conio.h>

int main()
{
    int j = -5; // initialization
    do
    {
        printf("%d\n", j);
        j = j + 1;
    }
    while(j <= 0);  // condition
    return 0;
}
```

Output:

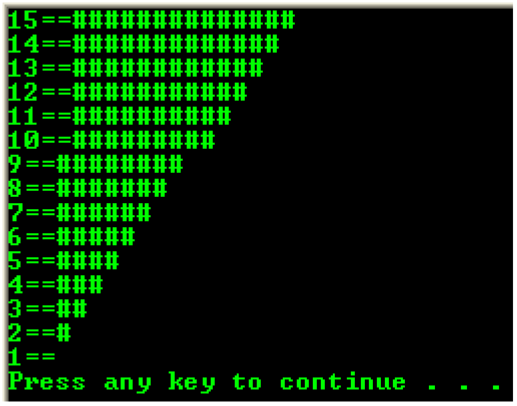


```
-5
-4
-3
-2
-1
0
Press any key to continue . . .
```

Example - 3:- Write a program that contains nested loops to draw a pyramid showing in output.

```
#include <stdio.h>
 int main()
{
    // variables for counter…
    int i = 15, j;
    // outer loop, execute this first...
    // for every i iteration, execute the inner loop
    while(i>0)
    {
        // display i==
        printf("%d==", i);
        // then, execute inner loop with loop index j,
        // the initial value of j is i - 1
        j=i-1;
        while(j>0)
        {
            // display #
            printf("#");
            // decrement j by 1 until j>10, i.e j = 9
            j = j - 1;
        }
        // go to new line, new row
        printf("\n");
        // decrement i by 1, repeat until i > 0 that is i = 1
        i = i - 1;
    }
    return 0;
}
```

Output:

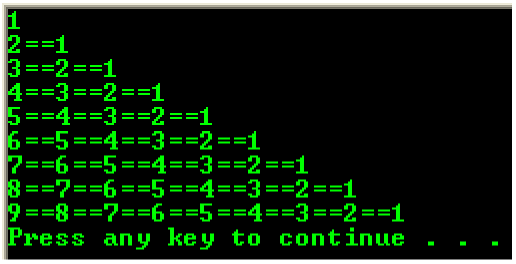


```
15==###############
14==##############
13==#############
12==############
11==###########
10==##########
9==#########
8==########
7==#######
6==######
5==#####
4==####
3==###
2==##
1==
Press any key to continue . . .
```

Example - 4:- Write a program that contains nested loops to display pyramid as shown in output.

```
# include <stdio.h>

# include<conio.h>

int main()

{

       // variables for counter…

       int i, j;

       // outer loop, execute this first...for every i iteration,

       // execute the inner loop

       for(i = 1; i <= 9;)

       {
            // display i

            printf("%d", i);

            // then, execute inner loop with loop index j,

            // the initial value of j is i - 1

            for(j = i-1; j>0; )

            {

                // display ==j

                printf("==%d", j);

                // decrement j by 1 until j>0, i.e j = 1

                j = j - 1;

            }

              // go to new line

              printf("\n");

              // increment i by 1, repeat until i<=9, i.e i = 9

              i = i + 1;

        }

        return 0;

}
```

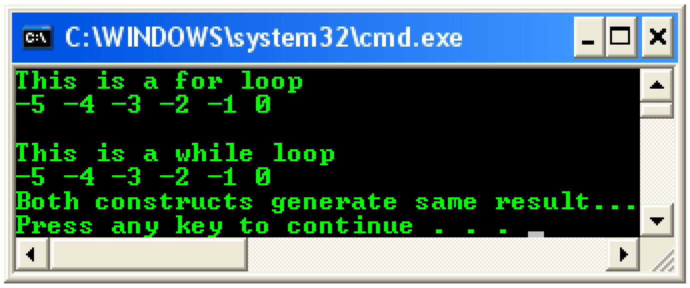
Output:



```
1
2==1
3==2==1
4==3==2==1
5==4==3==2==1
6==5==4==3==2==1
7==6==5==4==3==2==1
8==7==6==5==4==3==2==1
9==8==7==6==5==4==3==2==1
Press any key to continue . . .
```

Example - 5:- Write a program that display negative numbers from 0 to 5.

```
# include <stdio.h>

# include<conio.h>

int main(void)

{

    int i, j;

    // for loop

    printf("This is a for loop\n");

    for(i = -5; i <= 0; i = i +1) // initial, terminal condition and iteration

        printf("%d ", i);

    printf("\n");

    printf("\nThis is a while loop\n");

    j = -5; // initial condition

    // while loop

    while(j <= 0) // terminal condition

    {

        printf("%d ", j);

        j = j + 1;  // iteration

    }

    printf("\nBoth constructs generate same result...\n");

    return 0;

}
```



C:\WINDOWS\system32\cmd.exe

```
This is a for loop
-5 -4 -3 -2 -1 0

This is a while loop
-5 -4 -3 -2 -1 0
Both constructs generate same result...
Press any key to continue . . .
```

.

6.5 Exercises for lab

Exercises-1) Write a loop to do the following:

1. Read in 10 integers.

2. Find the maximum value.

3. Find the minimum value.

6.6 Home Work

1) Write a program that calculate the factorial of a number using while loop.

2) Write a program that inputs a number from a user and display its multiples using for loop.

3) Write a program that inputs a number and length from a user and display its table.