Programming Fundamentals

Week 7

Learning Outcomes:

- Students should be able to define repetitive statements
- Students should be able to repeat a Set of Instructions for a specific number of times using counter loop.
- Students should be able to repeat a Set of Instructions for an unknown number of times using conditional loop.
- Students should be able to solve larger problems by decomposing it into smaller sub-problems and combining their solution to achieve final results using the Counter Loop

Instructions

- Use proper indentation to make your programs readable.
- Use descriptive variables in your programs (Name of the variables should show their purpose)

Loop

Introduction

In real-world life, we are working with a repetitive process like bowler throwing six balls in an over, cash counting in banks, a wheel revolving its spindle. For this purpose, we need a structure that maps these methods into programming which is loops.

Loop:

A process which repeats itself again and again is called loop. In our daily life, we encounter many situations where we have to decide the repeated process.
For example

- Software of the ATM machine is in a loop to process transaction after transaction until you acknowledge that you have no more to do.
- Software program in a mobile device allows user to unlock the mobile with 5 password attempts.
- You put your favourite song on a repeat mode.

Before implementing loops, lets recall the concepts that you were taught in the earlier class.

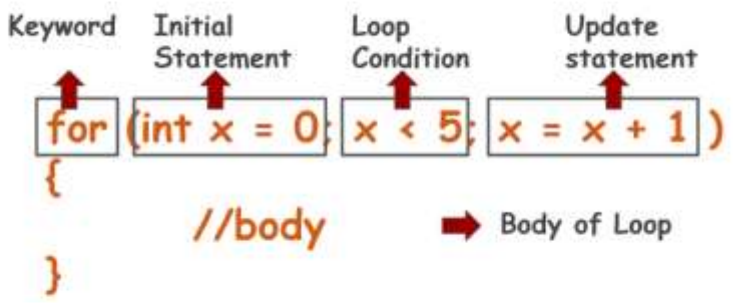
There are 2 types of Loops.

1. Counter Loops
2. Conditional Loops

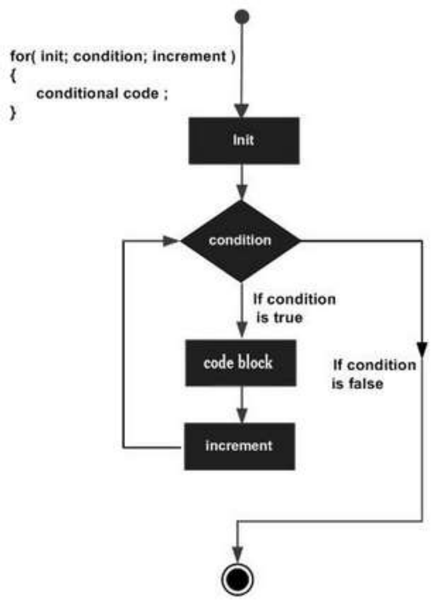
Counter Loop:

In Counter Loop, the program knows beforehand about how many times a specific instruction or set of instructions will be executed. In C++ programming language, **for loop** is used as a counter loop.

Syntax of For Loop:



Flow of For Loop:



Example #1:

Write a function show that display counting from 1 to 10.

Solution

```
#include <iostream>
using namespace std;
void show()
{
    for(int i = 1; i<=10; i= i+1)
    {
        cout << i << endl;
    }
}
main()
{
    show();
}
```

The code produces the following output

```
1
2
3
4
5
6
7
8
9
10
```

Explanation:

Iteration	Variable	i <= 10	Action
1st	`i = 1`	`True`	1 is printed. `i` is increased to `2`.
2nd	`i = 2`	`True`	2 is printed. `i` is increased to `3`.
3rd	`i = 3`	`True`	3 is printed. `i` is increased to `4`.
4th	`i = 4`	`True`	4 is printed. `i` is increased to `5`.
5th	`i = 5`	`True`	5 is printed. `i` is increased to `6`.
6th	`i = 6`	`True`	6 is printed. `i` is increased to `7`.
7th	`i = 7`	`True`	7 is printed. `i` is increased to `8`.
8th	`i = 8`	`True`	8 is printed. `i` is increased to `9`.
9th	`i = 9`	`True`	8 is printed. `i` is increased to `10`.
10th	`i = 10`	`True`	10 is printed. `i` is increased to `11`.
11th	`i = 11`	`false`	The loop is terminated

Example 1 was an example of Independent Iterations which means the result of previous iteration was not required for next iterations. However, there are set of problems that require the result from previous iterations. Such iterations are called dependent iterations.
Let’s see the Examples of Dependent Iterations.

Example #2:

Calculate the sum of first 5 natural numbers.

Remember: Before writing the program, first step is to identify the pattern within the sub problem such that this pattern can be written as a generic expression or set of generic expressions.

Solution

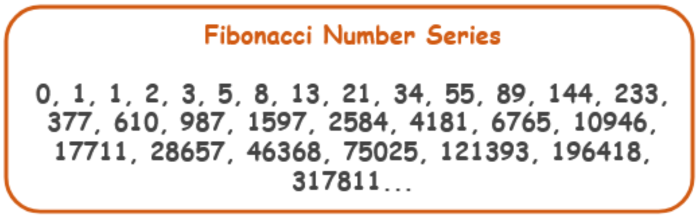
```cpp
#include <iostream>
#include<iostream>
using namespace std;
void showSum()
{
    int sum = 0;
    for(int i = 1; i<=5; i= i+1)
    {
        sum = sum + i;
    }
    cout << sum << endl;
}
main()
{
    showSum();
}
```

The code produces the following output

```
15
```

Example # 3:

Write a procedure that prompts the user to input length of Fibonacci series and display the series.



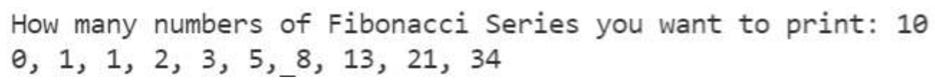
Fibonacci Number Series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811...

Solution

```cpp
#include <iostream>
using namespace std;
void fibonacciSeries(int leng)
{
    int n1 = 0, n2 = 1, next;
    cout << n1 << ", ";
    cout << n2;
    for(int x = 1; x < leng - 1; x = x + 1)
    {
        next = n1 + n2;
        cout << ", " << next;
        n1 = n2;
        n2 = next;
    }
}
main()
{
    int n;
    cout << "How many numbers of Fibonacci Series you want to print: ";
    cin >> n;
    fibonacciSeries(n);
}
```

The code produces the following output



```
How many numbers of Fibonacci Series you want to print: 10
0, 1, 1, 2, 3, 5,_8, 13, 21, 34
```

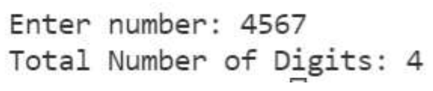
Example # 4:

Write a function named “total_Digits” that return total number of digits in a number.

Solution

```
#include<iostream>
using namespace std;
int total_digits(int number)
{
    int count = 0;
    for(int i = number; i>0; i= i/10)
    {
        count = count + 1;
    }
    return count;
}
main()
{
    int n, total;
    cout << "Enter number: ";
    cin >> n;
    total = total_digits(n);
    cout << "Total Number of Digits: " << total;

}
```

The code produces the following output



```
Enter number: 4567
Total Number of Digits: 4
```

Challenge#1:

Print multiplication table of 24, 50 and 29 using loop. Make a function whose prototype will be **void printTable(int number);**

Just call this function in main 3 times.

Challenge#2:

Find the frequency of a digit in a number. Make a function whose prototype will be

int frequencyChecker(int number, int digit);

you have to pass this function a number and a digit whose frequency you want to check then the function returns the number of times the digit occurs in the number.

Test Cases:

frequencyChecker(566960, 6) ⟶ **3**

frequencyChecker(566960, 5) ⟶ **1**

Congratulations, you have performed all the tasks using the For Loop. Now let's move towards conditional Loops.

Conditional Loop:

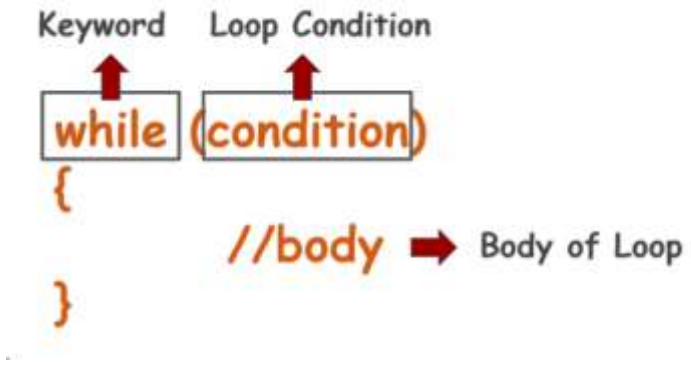
Conditional loops help to repeat a set of instructions until some condition is true. There are two common places for it use.

1. Reading an unknown amount of input from the user
2. Validating input.

C++ provides a while loop that is used as a conditional loop.

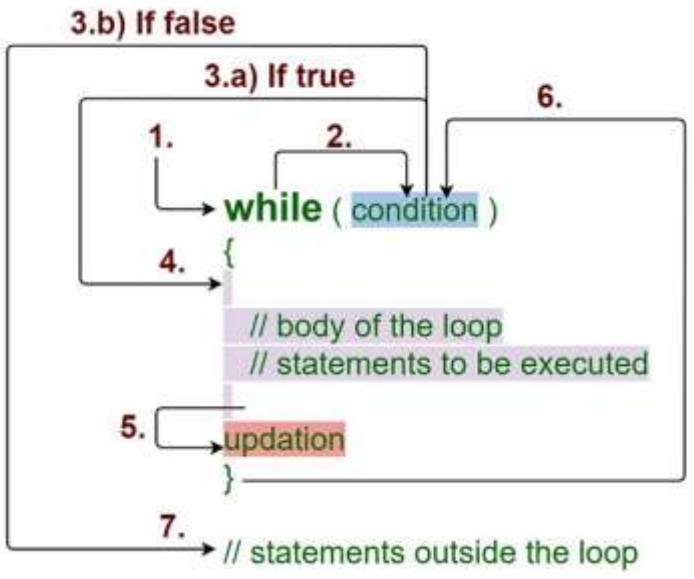
Syntax of While Loop

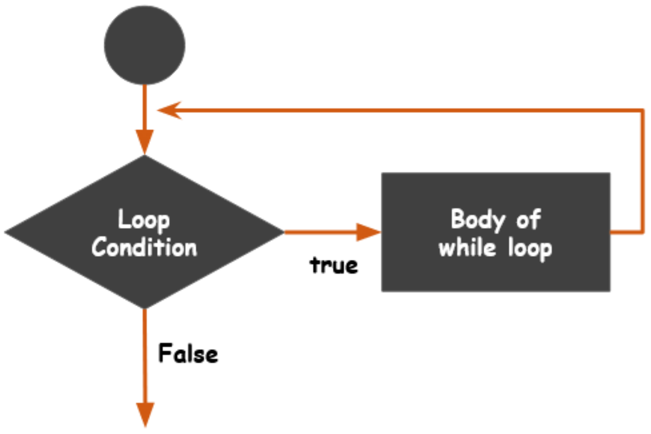
The syntax of a while loop in C++ is



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.

Flow of While Loop



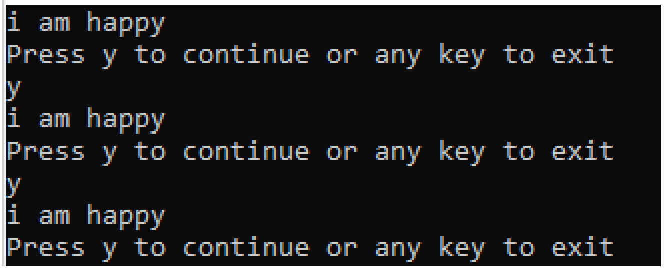


Example #1: **Print a message "I am happy" until user presses "y" or any key to exit.**

Solution

```
#include <iostream>
using namespace std;
void happyMessage()
{
    char ch = 'y';
    while(ch == 'y')
    {
        cout << "I am Happy" << endl;
        cout << "Press y to continue or any key to exit" << endl;
        cin >> ch;
    }
}
main()
{
    happyMessage();
}
```

The code produces the following output



```
i am happy
Press y to continue or any key to exit
y
i am happy
Press y to continue or any key to exit
y
i am happy
Press y to continue or any key to exit
```

Let's understand how we can validate input using a while loop with a working example.

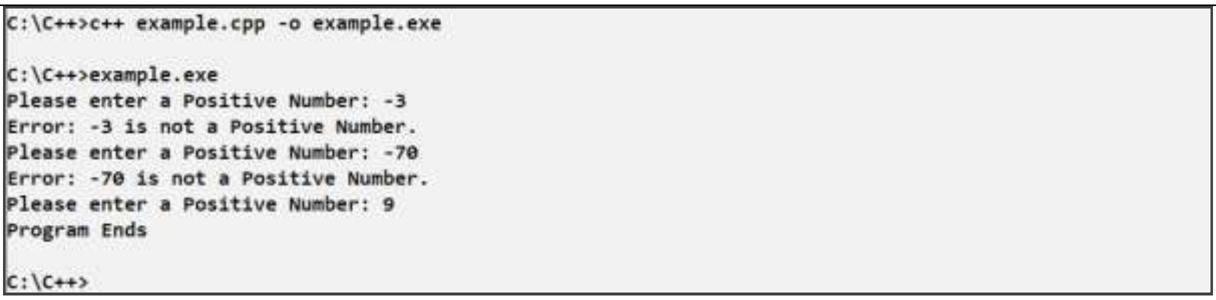
Example #2:

Suppose the requirement is to enter a positive number but if the user enters negative number then an error message is shown and the user is again asked to enter a positive number.

Solution

```
#include <iostream>
using namespace std;
void Validate()
{
    int value;
    cout << "Please enter a Positive Number: ";
    cin >> value;
    while (value <= 0)
    {
        cout << "Error: " << value << " is not a Positive Number." << endl;
        cout << "Please enter a Positive Number: ";
        cin >> value;
    }
}
main()
{
    Validate();
    cout << "Program Ends" << endl;
}
```

The code produces the following output



```
C:\C++>c++ example.cpp -o example.exe

C:\C++>example.exe
Please enter a Positive Number: -3
Error: -3 is not a Positive Number.
Please enter a Positive Number: -70
Error: -70 is not a Positive Number.
Please enter a Positive Number: 9
Program Ends

C:\C++>
```

There are many problems that you can solve with both for loop or while loop. It's up to you which loop you want to use in which you are more comfortable.

Let's see the first example of for loop.

Example # 3:

Write a function show that display counting from 1 to 10.

Solution with For Loop

```
#include <iostream>
using namespace std;
void show()
{
    for(int i = 1; i<=10; i= i+1)
    {
        cout << i << endl;
    }

}
main()
{
    show();
}
```

Solution with While Loop

```
#include <iostream>
using namespace std;
void show()
{
    int i = 1;
    while (i <= 10)
    {
        cout << i << endl;
        i = i + 1;
    }
}
main()
{
    show();
}
```

The code produces the following output

```
1
2
3
4
5
6
7
8
9
10
```

Example #4:

Calculate the sum of first 5 natural numbers.

Remember: Before writing the program, first step is to identify the pattern within the sub problem such that this pattern can be written as a generic expression or set of generic expressions.

Solution with For Loop

```
#include <iostream>
#include<iostream>
using namespace std;
void showSum()
{
    int sum = 0;
    for(int i = 1; i<=5; i= i+1)
    {
        sum = sum + i;
    }
    cout << sum << endl;
}
main()
{
    showSum();
}
```

Solution with While Loop

```
#include <iostream>
#include <iostream>
using namespace std;
void showSum()
{
    int i = 1, sum = 0;
    while (i <= 5)
    {
        sum = sum + i;
        i = i + 1;
    }
    cout << sum << endl;
}
main()
{
    showSum();
}
```

The code produces the following output

```
15
```

Congratulations, you have practiced and learned the fundamental concepts of For and While Loop. Let's start the challenges now. You can use any loop.

Challenge 1:

Write two separate functions to find greatest common divisor (GCD) or highest common factor (HCF) of given two numbers.

Greatest Common Divisor (GCD) or Highest Common Factor (HCF) of two positive integers is the largest positive integer that divides both numbers without remainder.

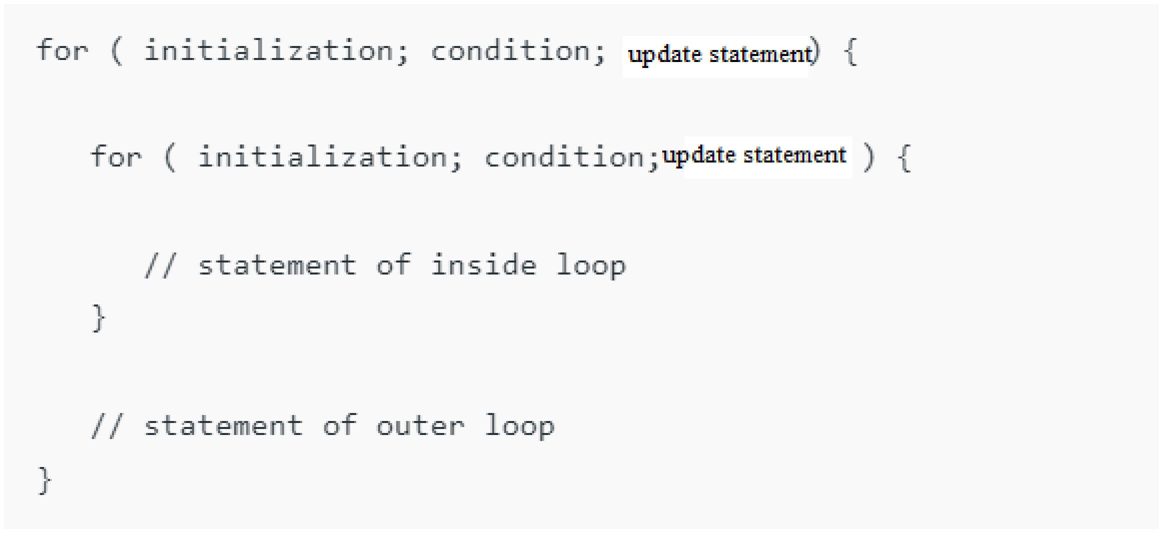
Least Common Multiple (LCM) of two integers is the smallest integer that is a multiple of both numbers.

Hint:

LCM(a, b) = (a * b) / GCD(a, b)

Nested For Loop:

A **nested loop** has one loop inside of another. These are typically used for working with two dimensions such as printing stars in rows and columns as shown below. When a loop is nested inside another loop, the inner loop runs many times inside the outer loop. In each iteration of the outer loop, the inner loop will be re-started. The inner loop must finish all of its iterations before the outer loop can continue to its next iteration.

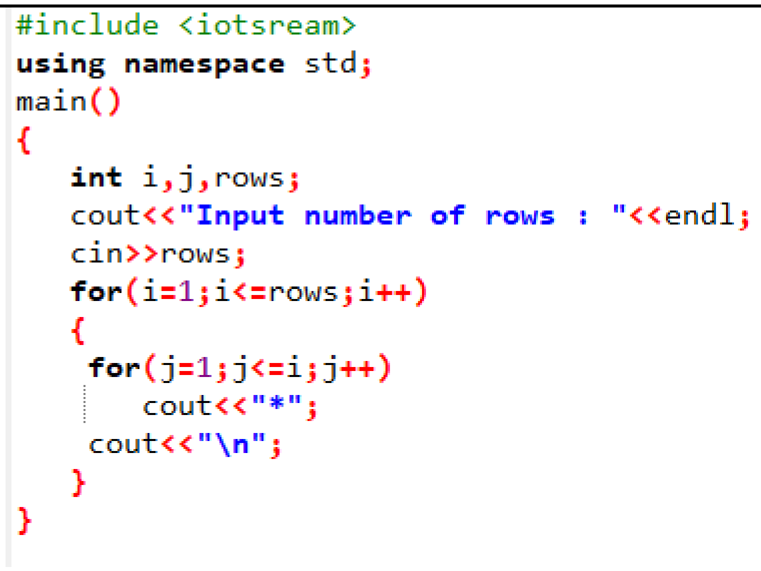


```
for ( initialization; condition; update statement) {

   for ( initialization; condition;update statement ) {

      // statement of inside loop
   }

   // statement of outer loop
}
```

Example #1 Write a program in C++ to display the pattern like right angle triangle using an asterisk.

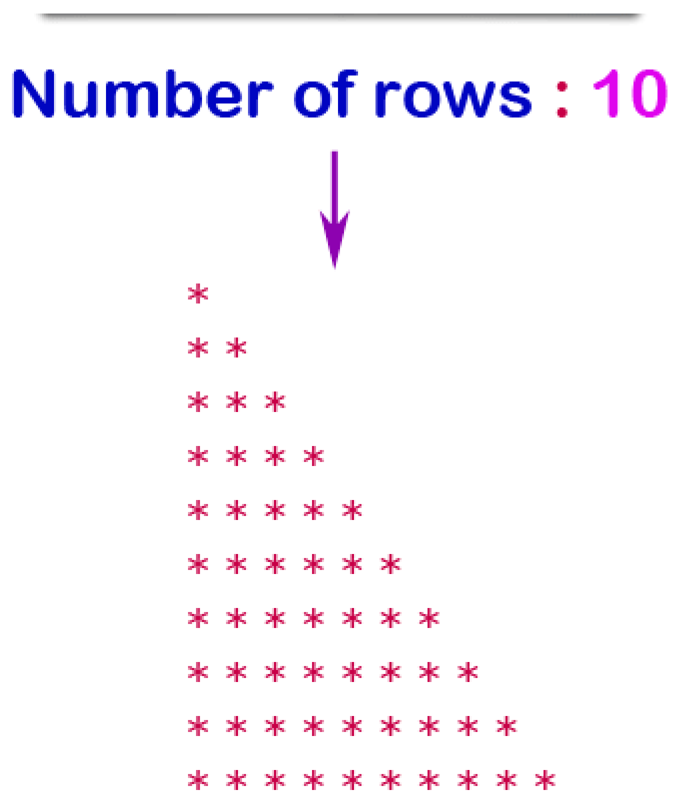
```
*
* *
* * *
* * * *
* * * * *
```

Solution
With Nested For



```cpp
#include <iotsream>
using namespace std;
main()
{
    int i,j,rows;
    cout<<"Input number of rows : "<<endl;
    cin>>rows;
    for(i=1;i<=rows;i++)
    {
     for(j=1;j<=i;j++)
         cout<<"*";
     cout<<"\n";
    }
}
```

The code produces the following output



```
Number of rows : 10

*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

The code produces the following output

Challenge#1: Write a program in C++ to display the pattern like diamond using an asterisk.

```
*
**
***
****
*****
*****
****
***
**
*
```