## Experiment No 3

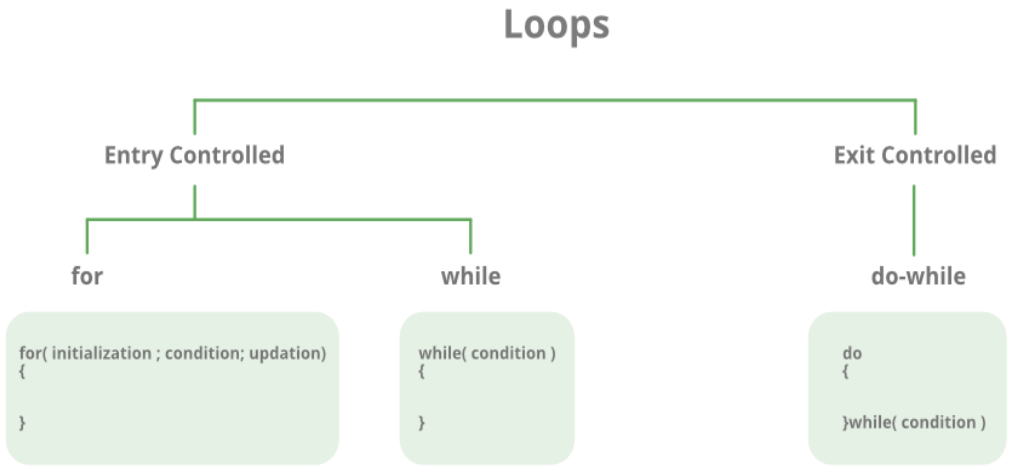
**Aim:** To implement the control statements in CPP.

**Theory:**

In Programming, sometimes there is a need to perform some operation **more than once** or (say) **n number** of times. Loops come into use when we need to repeatedly execute a block of statements.

**There are mainly two types of loops:**

1. **Entry Controlled loops**: In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loop** is entry-controlled loops.
2. **Exit Controlled Loops**: In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do-while **loop** is exit controlled loop.




**For Loop-**

A *For loop* is a repetition control structure that allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

**Syntax:**

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

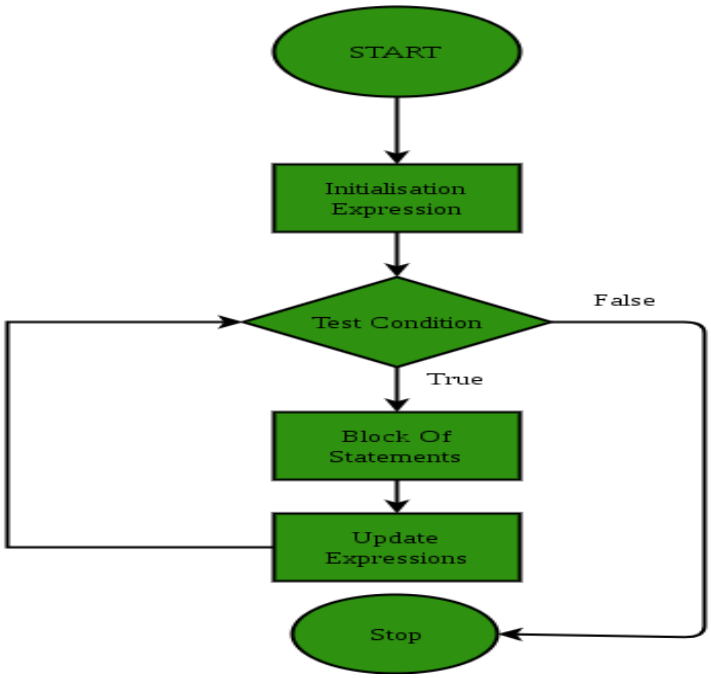
**Explanation of the Syntax:**

- **Initialization statement:** This statement gets executed only once, at the beginning of the for loop. You can enter a declaration of multiple variables of one type, such as int x=0, a=1, b=2. These variables are only valid in the scope of the loop. Variable defined before the loop with the same name are hidden during execution of the loop.
- **Condition:** This statement gets evaluated ahead of each execution of the loop body, and abort the execution if the given condition get false.

- **Iteration execution:** This statement gets executed after the loop body, ahead of the next condition evaluated, unless the for loop is aborted in the body (by break, goto, return or an exception being thrown.)

**NOTES:**

- The initialization and increment statements can perform operations unrelated to the condition statement, or nothing at all – if you wish to do. But the good practice is to only perform operations directly relevant to the loop.
- A variable declared in the initialization statement is visible only inside the scope of the for loop and will be released out of the loop.
- Don't forget that the variable which was declared in the initialization statement can be modified during the loop, as well as the variable checked in the condition.




**While Loop-**

While studying **for loop** we have seen that the number of iterations is ***known beforehand***, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where **we do not know** the exact number of iterations of the loop **beforehand**. The loop execution is terminated on the basis of the test conditions.

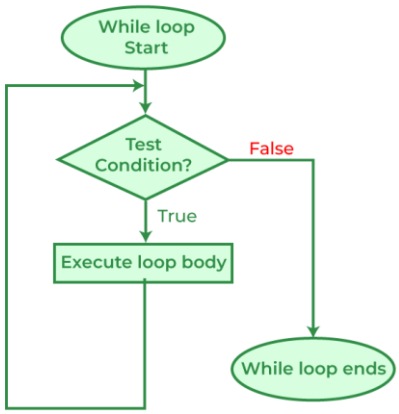
We have already stated that a loop mainly consists of three statements – initialization expression, test expression, and update expression. The syntax of the three loops – For, while, and do while mainly differs in the placement of these three statements.

**Syntax**:

```
initialization expression;
while (test_expression)
{
  // statements

 update_expression;
}
```

**Flow Diagram of while loop:**




**Do-while loop**

In Do-while loops also the loop execution is terminated on the basis of test conditions. The main difference between a do-while loop and the while loop is in the do-while loop the condition is tested at the end of the loop body, i.e do-while loop is exit controlled whereas the other two loops are entry-controlled loops.

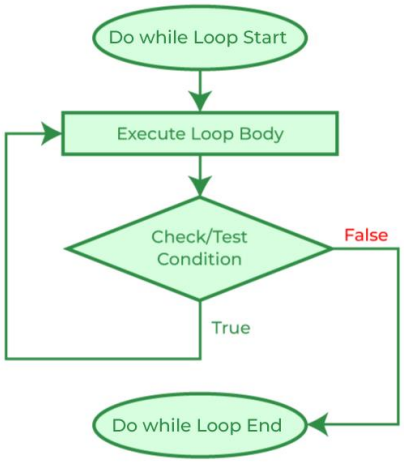
**Note**: In a do-while loop, the loop body will ***execute at least once*** irrespective of the test condition.

**Syntax**:

```
initialization expression;
do
{
  // statements
  update_expression;
} while (test_expression);
```

***Note**: Notice the semi – colon(";")in the end of loop.*

**Flow Diagram of the do-while loop:**



**Advantages :**

1. **High performance**: C++ is a compiled language that can produce efficient and high-performance code. It allows low-level memory manipulation and direct access to system resources, making it ideal for applications that require high performance, such as game development, operating systems, and scientific computing.
2. **Object-oriented programming:** C++ supports object-oriented programming, allowing developers to write modular, reusable, and maintainable code. It provides features such as inheritance, polymorphism, encapsulation, and abstraction that make code easier to understand and modify.
3. **Wide range of applications:** C++ is a versatile language that can be used for a wide range of applications, including desktop applications, games, mobile apps, embedded systems, and web development. It is also used extensively in the development of operating systems, system software, and device drivers.
4. **Standardized language:** C++ is a standardized language, with a specification maintained by the ISO (International Organization for Standardization). This ensures that C++ code written on one platform can be easily ported to another platform, making it a popular choice for cross-platform development.
5. **Large community and resources:** C++ has a large and active community of developers and users, with many resources available online, including documentation, tutorials, libraries, and frameworks. This makes it easy to find help and support when needed.
6. **Interoperability with other languages:** C++ can be easily integrated with other programming languages, such as C, Python, and Java, allowing developers to leverage the strengths of different languages in their applications.
   Overall, C++ is a powerful and flexible language that offers many advantages for developers who need to create high-performance, reliable, and scalable applications.

**Practical Related Questions:**

1. **Differentiate between while and do while loop.**
2. **Write the benefits of loops in programming language over conditional statements.**

**Programs :**

**Write a program in CPP**

1. Palindrome
2. Fibonacci Series
3. Armstrong number

**Conclusion :**

Hence, we learnt to implement the looping statements of CPP.

## PRACTICAL 3

**Write a Program in CPP to find whether the entered number is a palindrome or not.**

**Using For Loop**

```
#include <iostream>
using namespace std;
int main()
{
        int i,n,rev,rem,original;
        cout << "Enter an integer: ";
        cin >> n;
        original = n;
        // Reverse the integer
        for (i=n; i> 0; i/= 10)
        {
                rem = i % 10;
                rev = rev * 10 + rem;
        }
        cout << rev<<endl;
        // Check if the original and reversed numbers are equal
        if (original == rev)
                cout <<"\n"<< n << " is a palindrome.";
        else
                cout << n << " is not a palindrome.";
        return 0;
}
```

**Using While Loop**

```
#include <iostream>
using namespace std;
int main()
{
        int n, reversed = 0, remainder, original;
        cout << "Enter an integer: ";
        cin >> n;
        original = n;
        // Reverse the integer
        while (n != 0)
        {
                remainder = n % 10;
                reversed = reversed * 10 + remainder;
                n /= 10;
        }
   // Check if the original and reversed numbers are equal
   if (original == reversed)
      cout << n << " is a palindrome.";
   else
      cout << n << " is not a palindrome.";
   return 0;
```

```
}
```

**Using Do-While Loop**

```
#include <iostream>
using namespace std;
int main()
{
        int n, reversed = 0, remainder, original;
        cout << "Enter an integer: ";
        cin >> n;
        original = n;
        // Reverse the integer
        do
        {
            remainder = n % 10;
            reversed = reversed * 10 + remainder;
            n /= 10;
        } while (n != 0);
        // Check if the original and reversed numbers are equal
        if (original == reversed)
            cout << n << " is a palindrome.";
        else
            cout << n << " is not a palindrome.";
        return 0;
}
```

**Write a Program in CPP to print the Fibonacci series.**

**Using a For Loop**

```
#include <iostream>
using namespace std;
int main()
{
   int n, t1 = 0, t2 = 1, nextTerm;
        cout << "Enter the number of terms: ";
        cin >> n;
        cout << "Fibonacci Series: ";
        for (int i = 1; i<= n; ++i)
        {
            cout << t1 << ", ";
            nextTerm = t1 + t2;
            t1 = t2;
            t2 = nextTerm;
        }
        return 0;
}
```

**Using a While Loop**

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

```
int main()
{
        int n, t1 = 0, t2 = 1, nextTerm;
        cout << "Enter the number of terms: ";
        cin >> n;
        cout << "Fibonacci Series:";
        int i = 1;
        while (i <= n)
        {
                cout << t1 << ", ";
                nextTerm = t1 + t2;
                t1 = t2;
                t2 = nextTerm;
                i++;
        }
        return 0;
}
```

**Using a Do-While Loop**

```
#include <iostream>
using namespace std;
int main()
    {
        int n, t1 = 0, t2 = 1, nextTerm;
        cout << "Enter the number of terms: ";
        cin >> n;
        cout << "Fibonacci Series:";
        int i = 1;
        do
        {
                cout << t1 << ", ";
                nextTerm = t1 + t2;
                t1 = t2;
                t2 = nextTerm;
                i++;
        } while (i <= n);
        return 0;
}
```

**Write a Program in CPP to calculate Armstrong number of a entered number.**

**Using a For Loop**

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
        int num, originalNum, remainder, n = 0, sum = 0;
        cout << "Enter an integer: ";
        cin >> num;
```

```
    originalNum = num;
    // Find the number of digits
    while (originalNum != 0)
    {
            originalNum /= 10;
            ++n;
    }

    originalNum = num;
    // Calculate sum of power of digits
    for (int i = 0; i < n; ++i)
    {
            remainder = originalNum % 10;
            sum += pow(remainder, n);
            originalNum /= 10;
    }

    if (sum == num)
       cout << num << " is an Armstrong number";
    else
       cout << num << " is not an Armstrong number";

    return 0;
}
```

**Using a While Loop**

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int num, originalNum, remainder, n = 0, sum = 0;
    cout << "Enter an integer: ";
    cin >> num;
    originalNum = num;
    // Find the number of digits
    while (originalNum != 0)
    {
            originalNum /= 10;
            ++n;
    }
    originalNum = num;
    // Calculate sum of power of digits
    while (originalNum != 0)
    {
            remainder = originalNum% 10;
            sum += pow(remainder, n);
            originalNum /= 10;
    }
    if (sum == num)
       cout << num << " is an Armstrong number";
```

```
        else
            cout << num << " is not an Armstrong number";

        return 0;
}
```

**Using a Do-While Loop**

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
        int num, originalNum, remainder, n = 0, sum = 0;
        cout << "Enter an integer: ";
        cin >> num;
        originalNum = num;
        // Find the number of digits
        do
        {
                originalNum /= 10;
                ++n;
        } while (originalNum != 0);
        originalNum = num;
        // Calculate sum of power of digits
        do
        {
                remainder = originalNum % 10;
                sum += pow(remainder, n);
                originalNum /= 10;
        } while (originalNum != 0);
        if (sum == num)
            cout << num << " is an Armstrong number";
        else
            cout << num << " is not an Armstrong number";
        return 0;
}
```