2- Repetitively (Loop):

Loops in programming come into use when we need to repeatedly execute a block of statements. For example: Suppose we want to print “Hello World” 10 times. This can be done in two ways as shown below:

a- Iterative Method

An iterative method to do this is to write the cout << statement 10 times.

```
// C++ program to illustrate need of loops
#include <iostream>
using namespace std;

int main()
{
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      cout << "Hello World\n";
      return (0);
}
```

Output

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

b- Using Loops

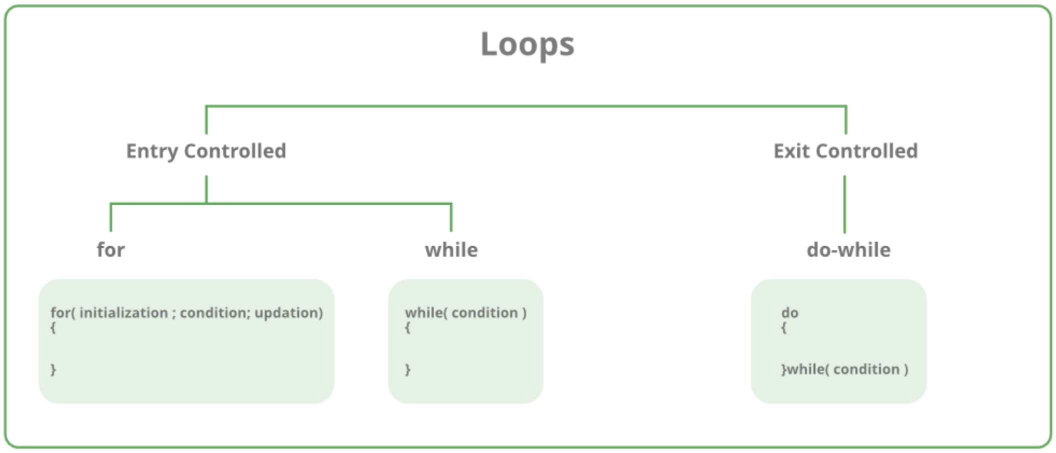
In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.

In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

- An operation is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.
- **Counter not Reached:** If the counter has not reached the desired number, the next instruction in the sequence returns to the first instruction in the sequence and repeat it.
- **Counter reached:** If the condition has been reached, the next instruction “falls through” to the next sequential instruction or branches outside the loop.

There are mainly two types of loops:

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



1- *for* Loop

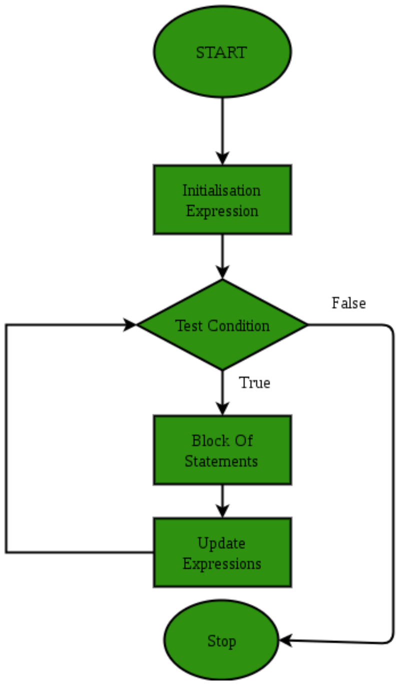
A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line. **Syntax:**

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

In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated . Steps are repeated till exit condition comes.

- **Initialization Expression**:
  In this expression we have to initialize the loop counter to some value. for example: int i=1;
- **Test Expression**:
  In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: i <= 10;
- **Update Expression**:
  After executing loop body this expression increments/decrements the loop variable by some value. for example: i++;

Flow diagram:



Example: C++ program to illustrate for loop

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

int main()
{
	for (int i = 1; i <= 10; i++)
	{
		cout << "Hello World\n";
	}

	return 0;
}
```

```
Output

Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

2- *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 loop beforehand. The loop execution is terminated on the basis of test condition.

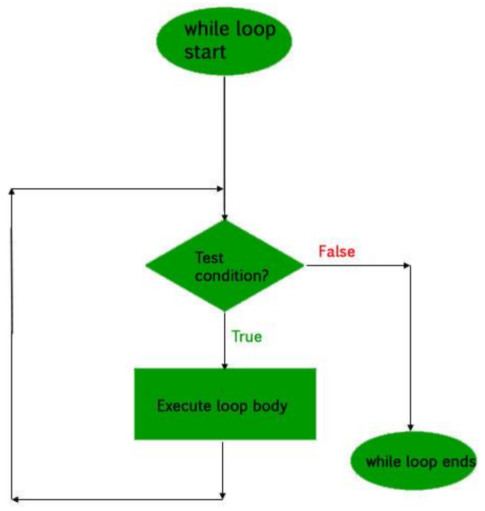
Syntax:

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

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

  update_expression;
}
```

Flow Diagram:



Example: C++ program to illustrate while loop

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

int main()
{
      // initialization expression
      int i = 1;

      // test expression
      while (i < 6)
      {
            cout << "Hello World\n";

            // update expression
            i++;
      }

      return 0;
}
```

```
Output

Hello World
Hello World
Hello World
Hello World
Hello World
```

3- *do while* loop

In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body, i.e do while loop is exit controlled whereas the other two loops are entry controlled loops. **Note**: In do while loop the loop body will execute at least once irrespective of test condition.

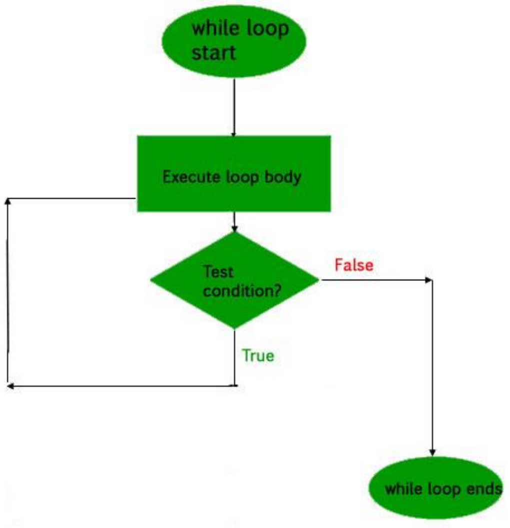
Syntax:

```
initialization expression;
do
{
   // statements

   update_expression;
} while (test_expression);
```

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

Flow Diagram:



Example: C++ program to illustrate do-while loop

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

int main(){
      int i = 1; // Initialization expression

      do
      {
            // loop body
            cout << "Hello World\n";

            // update expression
            i++;

      } while (i < =5); // test expression

      return 0;
}
```

```
Output

Hello World
Hello World
Hello World
Hello World
Hello World
```

What about an *Infinite* Loop?

An infinite loop (sometimes called an endless loop) is a piece of coding that lacks a functional exit so that it repeats indefinitely. An infinite loop occurs when a condition always evaluates to true. Usually, this is an error.

Example: C++ program to demonstrate infinite loops using for and while

```
// Uncomment the sections to see the output
#include <iostream>
using namespace std;
int main ()
{
int i;

// This is an infinite for loop as the condition expression is blank
for ( ; ; )
{
      cout << "This loop will run forever.\n";
}

// This is an infinite for loop as the condition
// given in while loop will keep repeating infinitely
/*
while (i != 0)
{
      i-- ;
      cout << "This loop will run forever.\n";
}
*/

// This is an infinite for loop as the condition
// given in while loop is "true"
/*
while (true)
{
      cout << "This loop will run forever.\n";
}
*/

}
```

```
Output:
This loop will run forever.
This loop will run forever.
...................
```

Nested loops
A loop can be nested inside of another loop. C++ allows at most 256 levels of nesting.

Example: C++ program uses a nested for loop to find the prime numbers from 2 to 100.

```
#include <iostream>
using namespace std;
int main () {
int i, j;
for (i = 2; i<= 100; i++)
{
      for (j = 2 ; j <= (i/j) ;  j++)
            if (!( I % j ))
                  break; // if factor found, not prime
            if (j > (i/j))
                  cout << i << " is prime\n";
}
return 0;
}
```

Important Points:

- Use for loop when number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known.
- Use while loops where exact number of iterations is not known but the loop termination condition is known.
- Use do while loop if the code needs to be executed at least once like in Menu driven programs.
- In summary: C++ programming language provides the following type of loops to handle looping requirements:

Loop Tool	Description
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
do...while loop	Like a ‘while’ statement, except that it tests the condition at the end of the loop body.
nested loops	You can use one or more loop inside any another ‘while’, ‘for’ or ‘do..while’ loop.

The goto statement

`goto` allows to make an absolute jump to another point in the program. This unconditional jump ignores nesting levels, and does not cause any automatic stack unwinding. Therefore, it is a feature to use with care, and preferably within the same block of statements, especially in the presence of local variables.
The destination point is identified by a *label*, which is then used as an argument for the `goto` statement. A *label* is made of a valid identifier followed by a colon (:).

`goto` is generally deemed a low-level feature, with no particular use cases in modern higher-level programming paradigms generally used with C++. But, just as an example, here is a version of our countdown loop using goto:

```
// goto loop example
#include <iostream>
using namespace std;

int main ()
{
  int n=10;
mylabel:
  cout << n << ", ";
  n--;
  if (n>0) goto mylabel;
  cout << "liftoff!\n";
  return (0)
}
```

Output

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!
```

More Advanced Looping Techniques

- Range-based for loop in C++
- for_each loop in C++

Jumping out of a loop

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becocmes true, that is jump out of loop. C language allows jumping from one statement to another within a loop as well as jumping out of the loop.

1) break statement

`break` leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end. For example, let's stop the countdown before its natural end:

```
// break loop example
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--)
  {
    cout << n << ", ";
    if (n==3)
    {
      cout << "countdown aborted!";
      break;
    }
  }
Return (0);
}
```

```
Output

10, 9, 8, 7, 6, 5, 4, 3, countdown aborted!
```

2) continue statement

The `continue` statement causes the program to skip the rest of the loop in the current iteration, as if the end of the statement block had been reached, causing it to jump to the start of the following iteration. For example, let's skip number 5 in our countdown:

```
// continue loop example
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5)
        continue;
    cout << n << ", ";
  }
  cout << "liftoff!\n";
return (0);
}
```

```
Output

10, 9, 8, 7, 6, 4, 3, 2, 1, liftoff!
```