There are three different types of loop in C++ language: ***for***, ***while***, and ***do..while***.

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

# Syntax:

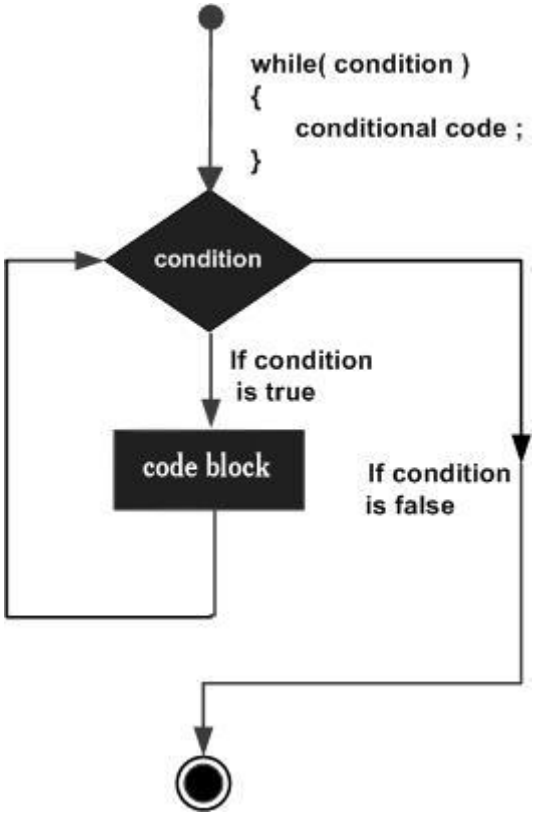
The syntax of a while loop in C++ is:

```
while(condition)
{
   statement(s);
}
```

Here **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero 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 Diagram:




Here key point of the *while* loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

# Example:

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

int main ()
```

```
{
   // Local variable declaration:
   int a = 10;

   // while loop execution
   while( a < 20 )
   {
       cout << "value of a: " << a << endl;
       a++;
   }

   return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

# Syntax:

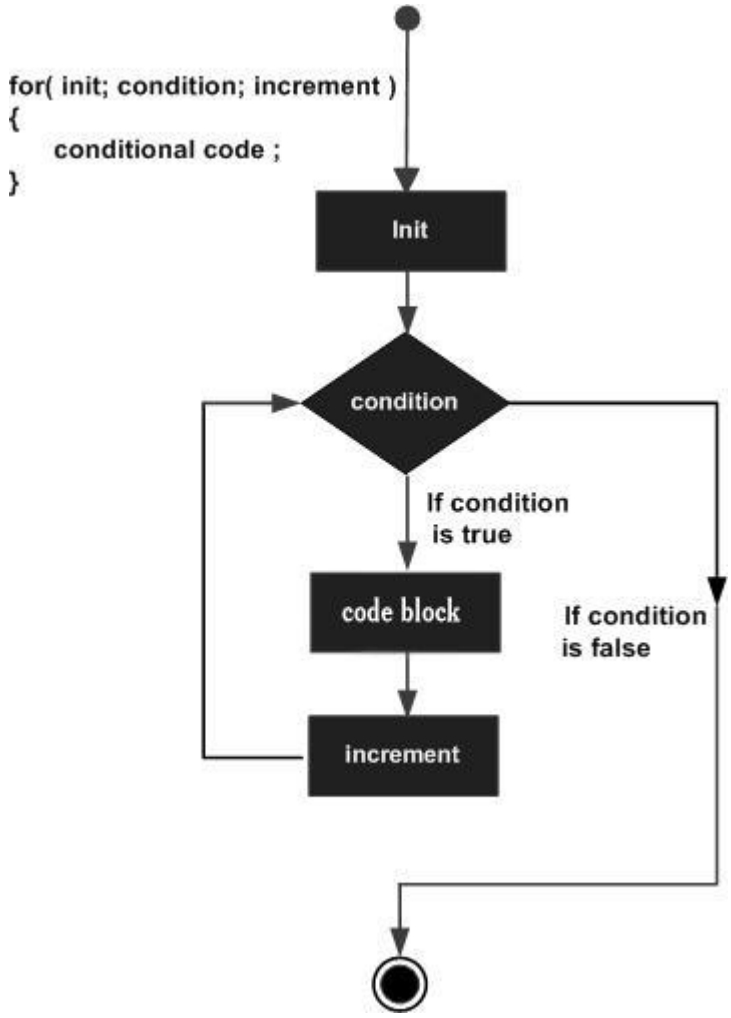
The syntax of a for loop in C++ is:

```
for ( init; condition; increment )
{
   statement(s);
}
```

Here is the flow of control in a for loop:

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the **increment**statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

# Flow Diagram:




# Example:

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

int main ()
{
   // for loop execution
   for( int a = 10; a < 20; a = a + 1 )
   {
       cout << "value of a: " << a << endl;
   }

   return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
```

```
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.
A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

# Syntax:

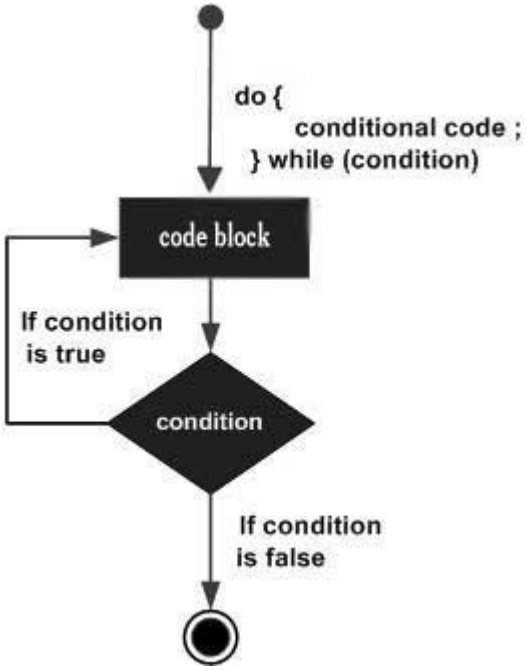
The syntax of a do...while loop in C++ is:

```
do
{
   statement(s);
}while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

# Flow Diagram:




# Example:

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

int main ()
{
   // Local variable declaration:
   int a = 10;

   // do loop execution
   do
```

```
   {
       cout << "value of a: " << a << endl;
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```