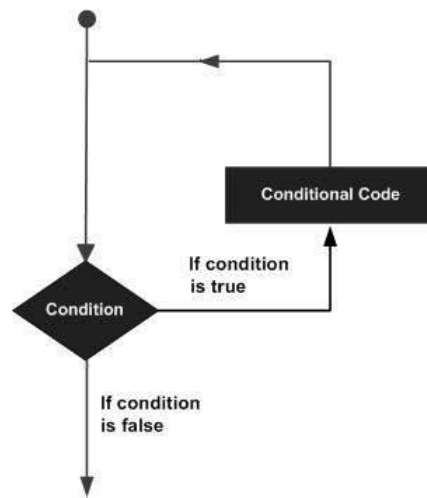


C++ Loop Types

There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –



C++ programming language provides the following type of loops to handle looping requirements.

Sr.No	Loop Type & Description
1	<u>while loop</u> Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	<u>for loop</u> Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	<u>do...while loop</u> Like a 'while' statement, except that it tests the condition at the end of the loop body.
4	<u>nested loops</u> You can use one or more loop inside any another 'while', 'for' or 'do..while' loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C++ supports the following control statements.

Sr.No	Control Statement & Description
1	<u>break statement</u> Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
2	<u>continue statement</u> Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	<u>goto statement</u> Transfers control to the labeled statement. Though it is not advised to use goto statement in your program.

The Infinite Loop

A loop becomes infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

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

int main () {
    for( ; ; ) {
        printf("This loop will run forever.\n");
    }

    return 0;
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C++ programmers more commonly use the 'for (;;)' construct to signify an infinite loop.

NOTE – You can terminate an infinite loop by pressing Ctrl + C keys.

C++ while loop

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

Syntax

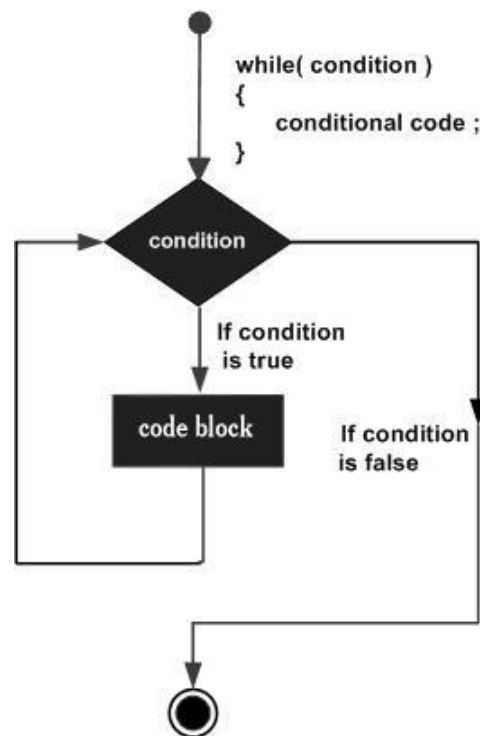
The syntax of a while loop in C++ is –

```
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 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 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 the 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
```

C++ for loop

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

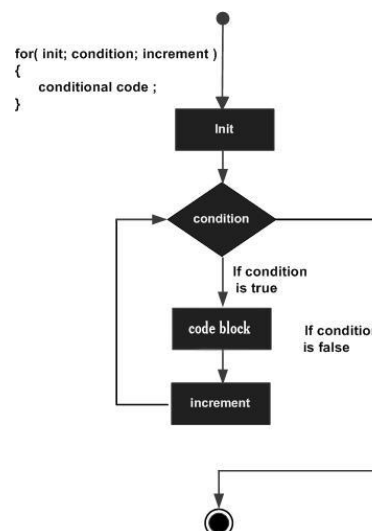
The syntax of a for loop in C++ is –

```
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 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 the 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
```

C++ do...while loop

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

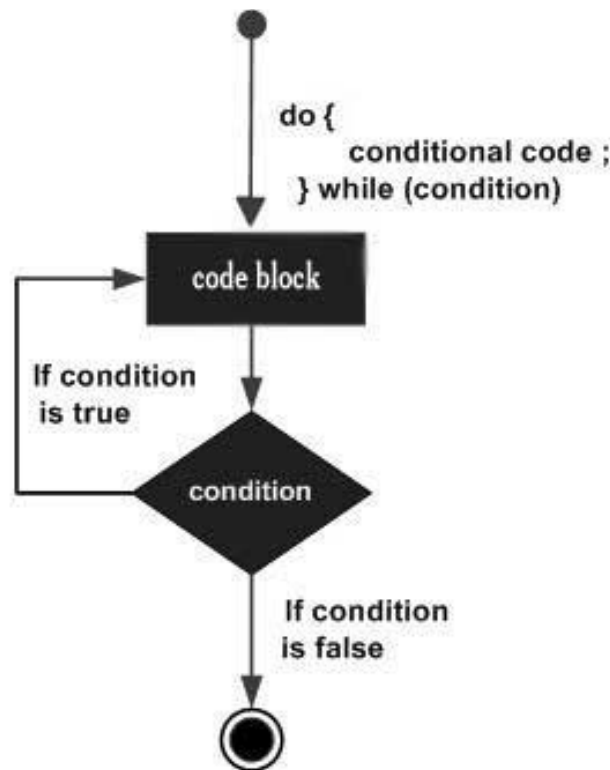
The syntax of a do...while loop in C++ is –

```
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 the 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
```