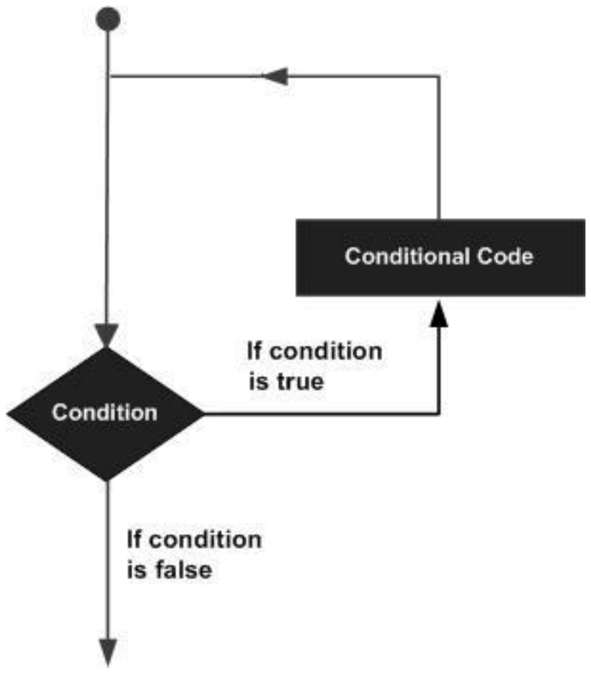
LOOPS IN C

You may encounter situations, when a block of code needs to be executed 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. Given below is the general form of a loop statement in most of the programming languages −



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

Sr.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop

	It is more like a while statement, except that it tests the condition at the end of the loop body.
4	nested loops You can use one or more loops inside any other 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	break statement Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	goto statement Transfers control to the labeled statement.

The Infinite Loop

A loop becomes an 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 <stdio.h>

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.

for Loop

The syntax of the `for` loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
    // statements inside the body of loop
}
```

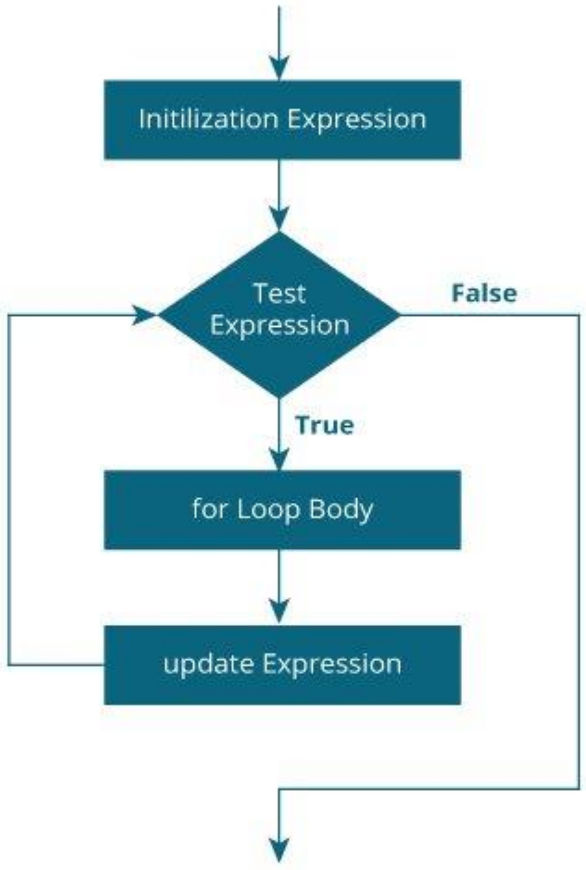
How for loop works?

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of the for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.

This process goes on until the test expression is false. When the test expression is false, the loop terminates.

To learn more about test expression (when the test expression is evaluated to true and false), check out relational and logical operators.

for loop Flowchart



Working of for loop

Example 1: for loop

```
// Print numbers from 1 to 10
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

Output

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

1. `i` is initialized to 1.
2. The test expression `i < 11` is evaluated. Since 1 less than 11 is true, the body of `for` loop is executed. This will print the **1** (value of `i`) on the screen.
3. The update statement `++i` is executed. Now, the value of `i` will be 2. Again, the test expression is evaluated to true, and the body of `for` loop is executed. This will print **2** (value of `i`) on the screen.
4. Again, the update statement `++i` is executed and the test expression `i < 11` is evaluated. This process goes on until `i` becomes 11.
5. When `i` becomes 11, `i < 11` will be false, and the `for` loop terminates.

Example 2: for loop

```
// Program to calculate the sum of first n natural numbers
// Positive integers 1,2,3...n are known as natural numbers

#include <stdio.h>
int main()
{
    int num, count, sum = 0;

    printf("Enter a positive integer: ");
    scanf("%d", &num);

    // for loop terminates when num is less than count
    for(count = 1; count <= num; ++count)
    {
        sum += count;
    }

    printf("Sum = %d", sum);

    return 0;
}
```

Output

```
Enter a positive integer: 10
Sum = 55
```

- The value entered by the user is stored in the variable `num`. Suppose, the user entered 10.
- The `count` is initialized to 1 and the test expression is evaluated. Since the test expression `count<=num` (1 less than or equal to 10) is true, the body of `for` loop is executed and the value of `sum` will equal to 1.
- Then, the update statement `++count` is executed and `count` will equal to 2. Again, the test expression is evaluated. Since 2 is also less than 10, the test expression is evaluated to true and the body of the `for` loop is executed. Now, `sum` will equal 3.
- This process goes on and the sum is calculated until the `count` reaches 11.
- When the `count` is 11, the test expression is evaluated to 0 (false), and the loop terminates.
- Then, the value of `sum` is printed on the screen.

while and do...while Loop

In this tutorial, you will learn to create while and do...while loop in C programming with the help of examples.

while loop

The syntax of the `while` loop is:

```
while (testExpression) {
  // the body of the loop
}
```

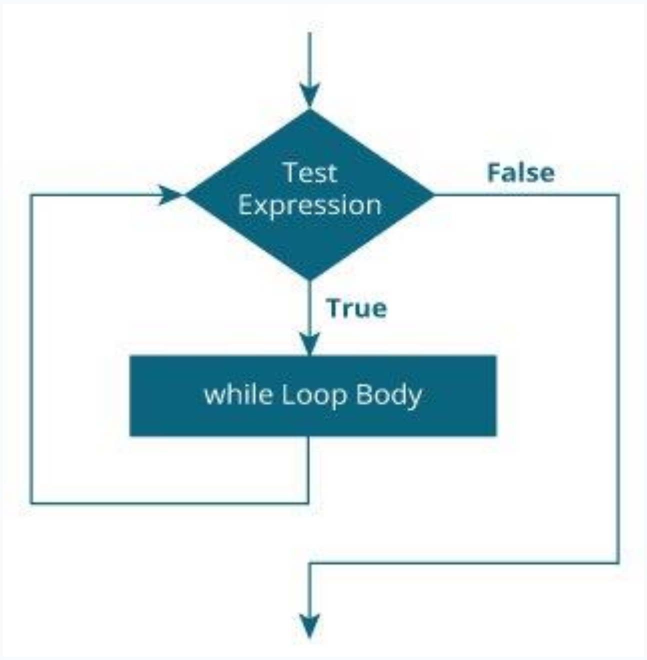
How while loop works?

- The `while` loop evaluates the `testExpression` inside the parentheses `()`.

- If testExpression is **true**, statements inside the body of while loop are executed.

  Then, testExpression is evaluated again.
- The process goes on until testExpression is evaluated to **false**.
- If testExpression is **false**, the loop terminates (ends).

Flowchart of while loop



Example 1: while loop

```
// Print numbers from 1 to 5

#include <stdio.h>
int main() {
  int i = 1;

  while (i <= 5) {
    printf("%d\n", i);
    ++i;
  }

  return 0;
}
```
Run Code

Output

```
1
2
```

```
3
4
5
```

Here, we have initialized i to 1.

1. When i = 1, the test expression i <= 5 is **true**. Hence, the body of the while loop is executed. This prints 1 on the screen and the value of i is increased to 2.
2. Now, i = 2, the test expression i <= 5 is again **true**. The body of the while loop is executed again. This prints 2 on the screen and the value of i is increased to 3.
3. This process goes on until i becomes 6. Then, the test expression i <= 5 will be **false** and the loop terminates.

do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

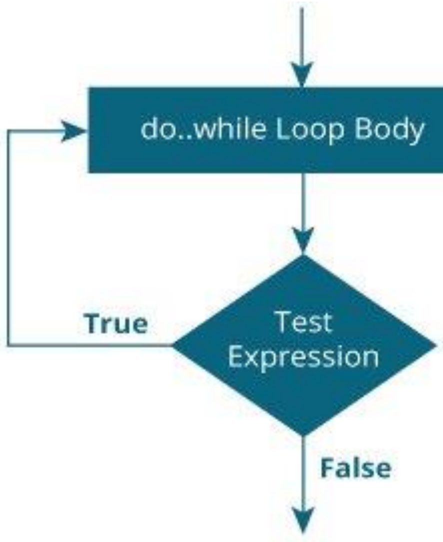
The syntax of the do...while loop is:

```
do {
  // the body of the loop
}
while (testExpression);
```

How do...while loop works?

- The body of do...while loop is executed once. Only then, the testExpression is evaluated.
- If testExpression is **true**, the body of the loop is executed again and testExpression is evaluated once more.
- This process goes on until testExpression becomes **false**.
- If testExpression is **false**, the loop ends.

Flowchart of do...while Loop



Working of do...while loop

Example 2: do...while loop

```
// Program to add numbers until the user enters zero

#include <stdio.h>
int main() {
  double number, sum = 0;

  // the body of the loop is executed at least once
  do {
    printf("Enter a number: ");
    scanf("%lf", &number);
    sum += number;
  }
  while(number != 0.0);

  printf("Sum = %.2lf",sum);

  return 0;
}
```
Run Code

Output

```
Enter a number: 1.5
Enter a number: 2.4
Enter a number: -3.4
Enter a number: 4.2
Enter a number: 0
Sum = 4.70
```

- Here, we have used a `do...while` loop to prompt the user to enter a number. The loop works as long as the input number is not `0`.
- The `do...while` loop executes at least once i.e. the first iteration runs without checking the condition. The condition is checked only after the first iteration has been executed.

```
do {
  printf("Enter a number: ");
  scanf("%lf", &number);
  sum += number;
}
while(number != 0.0);
```

- So, if the first input is a non-zero number, that number is added to the `sum` variable and the loop continues to the next iteration. This process is repeated until the user enters `0`.
- But if the first input is 0, there will be no second iteration of the loop and `sum` becomes `0.0`.
- Outside the loop, we print the value of `sum`.