C - Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1) It provides code reusability.

2) Using loops, we do not need to write the same code again and again.

3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

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

1. for loop
2. while loop
3. do...while loop
4. nested loops

for Loop

The syntax of the `for` loop is:

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

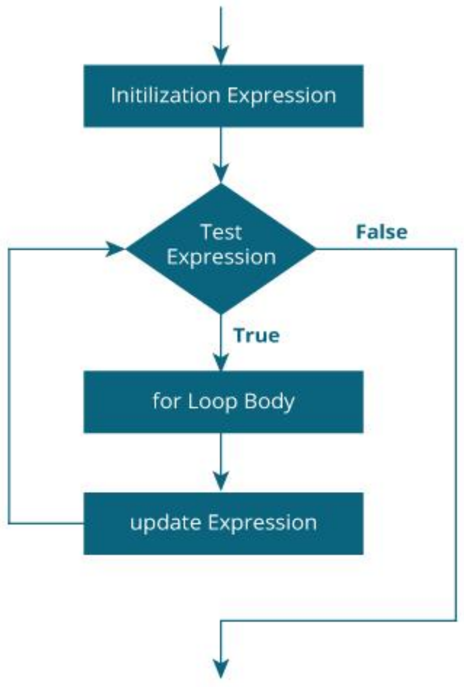
How for loop works?

- 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 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.

for loop Flowchart



```
Example 1: 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: Program to calculate the sum of first n 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 the 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 `for` loop is executed. Now, the *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 loop

The syntax of the `while` loop is:

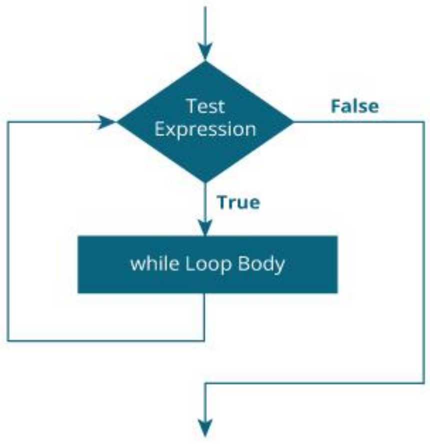
```
1. while (testExpression)
```

```
{
    // statements inside the body of the loop
}
```

How while loop works?

- The while loop evaluates the test expression inside the parenthesis ().
- If the test expression is true, statements inside the body of while loop are executed. Then, the test expression is evaluated again.
- The process goes on until the test expression is evaluated to false.
- If the test expression is false, the loop terminates (ends).

Flowchart of while loop



```
Example 1: Print numbers from 1 to 5
  #include <stdio.h>
  int main()
  {
      int i = 1;

      while (i <= 5)
      {
          printf("%d\n", i);
          ++i;
      }
      return 0;
  }
```

Output

```
1
2
3
4
5
```

Here, we have initialized *i* to 1.

1. When *i* is 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* is 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. When *i* is 6, 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.

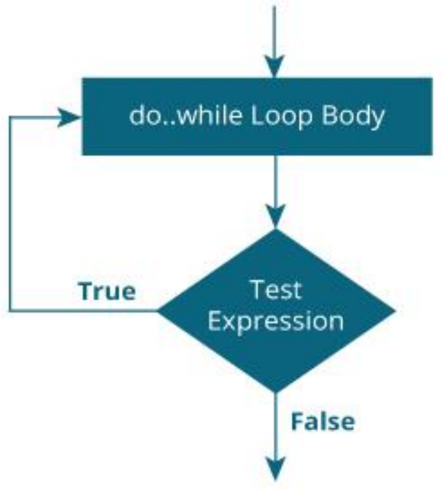
The syntax of the `do...while` loop is:

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

How do...while loop works?

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

Flowchart of do...while Loop



```
Example 1:   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;
    }
```

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
```

Try the following programs:

1. C program to print ODD numbers from 1 to N using while loop.
2. C program to print EVEN numbers from 1 to N using while loop.
3. C program to print all uppercase alphabets using while loop.
4. C program to print all lowercase alphabets using while loop.
5. C program to print numbers from 1 to N using while loop.
6. C program to print numbers from 1 to 10 using while loop.
7. C program to read an integer and print its multiplication table.
8. C Program to print tables from numbers 1 to 20.
9. C Program to check entered number is ZERO, POSITIVE or NEGATIVE until user does not want to quit.
10. C Program to find factorial of a number.
11. C Program to find sum of first N natural number, N must be taken by the user.
12. C program to print all prime numbers from 1 to N.
13. C program to print all even and odd numbers from 1 to N.
14. C program to print all Armstrong numbers from 1 to N.
15. C program to print square, cube and square root of all numbers from 1 to N.
16. C program to print all leap years from 1 to N.
17. C program to print all upper case and lower case alphabets.
18. C program to read age of 15 persons and count total Baby age, School age and Adult age.