# Decision making and looping:

In looping, a sequence of statements is executed until some conditions for termination of the loop are satisfied. A program loop therefore consists of two segments, one known as the body of the loop and other known as the control statement. The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

Depending on the position of the control statement in the loop, a control structure may be classified either as the entry controlled loop or as the exit controlled loop.

What is the **Difference between Entry Controlled and Exit Controlled Loop** in C?

## Entry and Exit Controlled Loop in C

Loops are the technique to repeat set of statements until given condition is true. C programming language has three types of loops **- 1) while loop**, **2) do while loop**and **3) for loop**.
These loops controlled either at entry level or at exit level hence loops can be controlled two ways

1. Entry Controlled Loop
2. Exit Controlled Loop

### Entry Controlled Loop

Loop, **where test condition is checked before entering the loop body**, known as **Entry Controlled Loop**.
**Example: while** loop, **for** loop

### Exit Controlled Loop

Loop, **where test condition is checked after executing the loop body**, known as **Exit Controlled Loop**.
**Example: do while** loop

Consider the code snippets

**Using while loop**

```
int count=100;
while(count<50)
   printf("%d",count++);
```

**Using for loop**

```
int count;
for(count=100; count<50; count++)
   printf("%d",count);
```

In both code snippets **value of count is 100** and condition is **count<50**, which will check first, hence there is no output.

**Using do while loop**

```
int count=100;
do
{
   printf("%d",count++);
}while(count<50);
```

In this code snippet **value of count is 100** and test condition is **count<50** which is false yet loop body will be executed first then condition will be checked after that. Hence output of this program will be 100.

### Differences Table

| **Entry Controlled Loop** | **Exit Controlled Loop** |
|---|---|

| | |
|---|---|
| Test condition is checked first, and then loop body will be executed. | Loop body will be executed first, and then condition is checked. |
| If Test condition is false, loop body will not be executed. | If Test condition is false, loop body will be executed once. |
| for loop and while loop are the examples of Entry Controlled Loop. | do while loop is the example of Exit controlled loop. |
| Entry Controlled Loops are used when checking of test condition is mandatory before executing loop body. | Exit Controlled Loop is used when checking of test condition is mandatory after executing the loop body. |

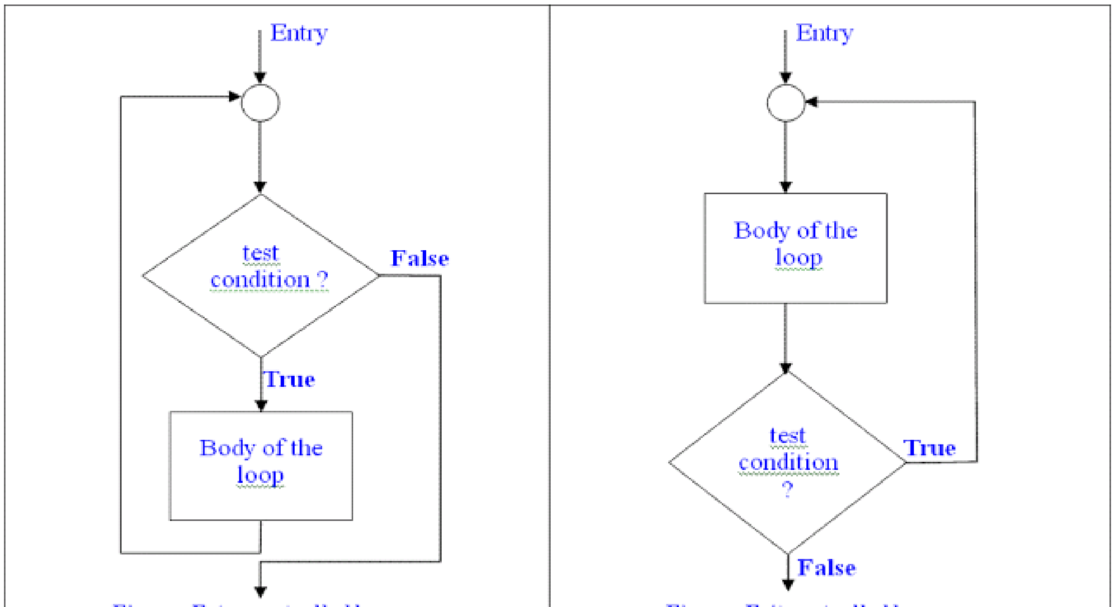


Figure : Entry controlled loop

Figure : Exit controlled loop

There are three loops in C programming:

1. for loop
2. while loop
3. do...while loop

Based on the nature of the control variables and the kind of value assigned to, the loops may be classified into two general categories;
->> counter controlled and
->>sentinel controlled loops.

**Counter controlled loops :** The type of loops, where the number of the execution is known in advance are termed by the counter controlled loop. That means, in this case, the value of the variable which controls the execution of the loop is previously known. The control variable is known as counter. A counter controlled loop is also called definite repetition loop.

**Example :** A while loop is an example of counter controlled loop.

```
= = = = = = = = =
sum = 0;
n = 1;
while (n <= 10)
{
sum = sum + n*n;
n = n+ 1;
}
= = = = = = = = =
```

This is a typical example of counter controlled loop. Here, the loop will be executed exactly 10 times for n = 1,2,3,......,10.

**Sentinel controlled loop :** The type of loop where the number of execution of the loop is unknown, is termed by sentinel controlled loop. In this case, the value of the control variable differs within a limitation and the execution can be terminated at any moment as the value of the variable is not controlled by the loop. The control variable in this case is termed by sentinel variable.

**Example :** The following **do....while** loop is an example of sentinel controlled loop.

```
    = = = = = =
do
{
printf(“Input a number.\n”);
scanf("%d", &num);
}
while(num>0);
= = = = = =
```

In the above example, the loop will be executed till the entered value of the variable **num** is not 0 or less then 0. This is a sentinel controlled loop and here the variable **num** is a sentinel variable.

## while loop:

The syntax of a while loop is:
while (testExpression)

```
{
   //codes
}
```

**How while loop works?**

The while loop evaluates the test expression.
If the test expression is true (nonzero), codes inside the body of while loop is evaluated. Then, again the test expression is evaluated. The process goes on until the test expression is false. When the test expression is false, the while loop is terminated.

Flowchart of while loop

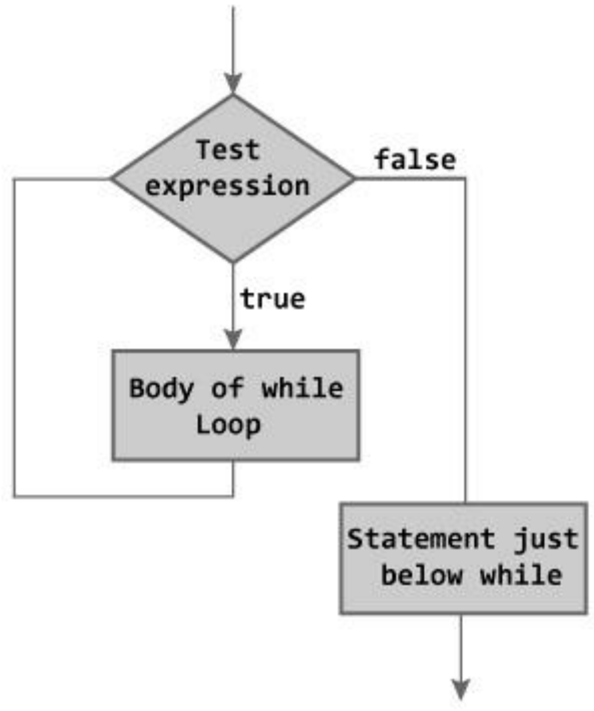


Figure: Flowchart of while Loop

Example #1: while loop

```
// Program to find factorial of a number
// For a positive integer n, factorial = 1*2*3...n

#include <stdio.h>
int main()
{
    int number;
    long long factorial;
    printf("Enter an integer: ");
    scanf("%d",&number);
    factorial = 1;
    // loop terminates when number is less than or equal to 0
    while (number > 0)
    {
        factorial *= number;  // factorial = factorial*number;
        --number;
    }
    printf("Factorial= %lld", factorial);
    return 0;
}
```

Output

```
Enter an integer: 5
Factorial = 120
```

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out relational and logical operators.

## 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 once, before checking the test expression. Hence, the do...while loop is executed at least once.

### do...while loop Syntax

```
do{
// codes
}while (testExpression);
```

### How do...while loop works?

The code block (loop body) inside the braces is executed once.Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false).
When the test expression is false (nonzero), the do...while loop is terminated.

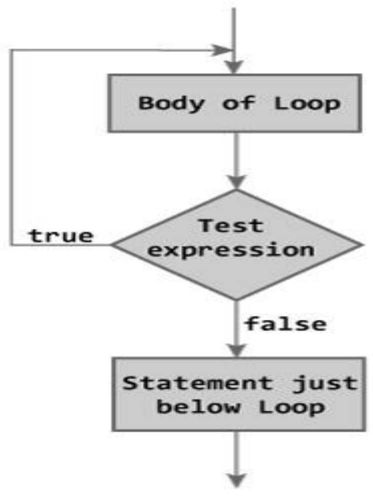


Figure: Flowchart of do...while Loop

### Example #2: do...while loop

```
// Program to add numbers until user enters zero
#include <stdio.h>
int main()
{
    double number, sum = 0;
    // loop body 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
```

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out relational and logical operators.

## for Loop

The syntax of a for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
      // codes
}
```

## How for loop works?

The initialization statement is executed only once. Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and update expression is updated. This process repeats until the test expression is false.The for loop is commonly used when the number of iterations is known. To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out relational and logical operators.

## for loop Flowchart

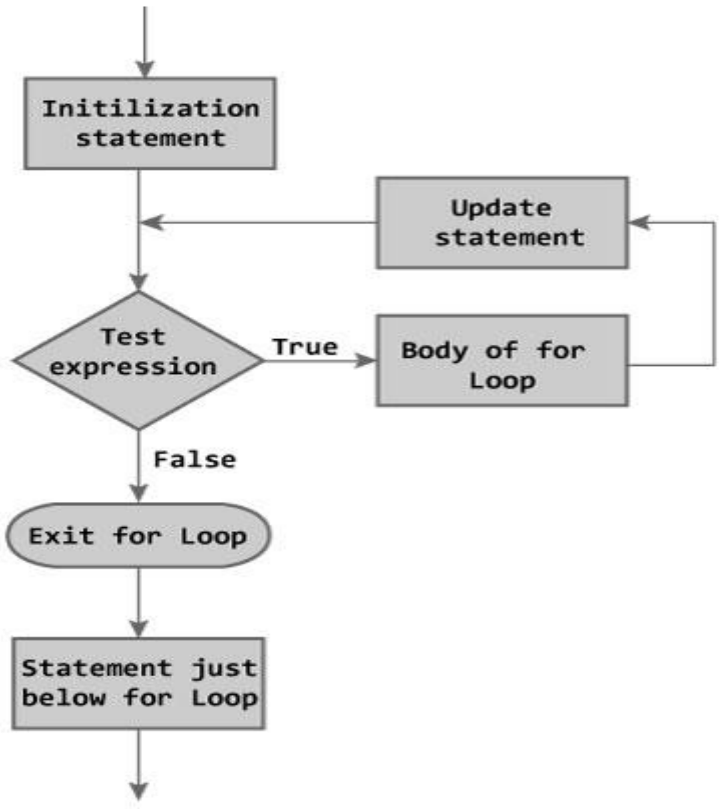


Figure: Flowchart of for Loop

## Example: 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 n, count, sum = 0;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    // for loop terminates when n is less than count
    for(count = 1; count <= n; ++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 variable n. Suppose the user entered 10.
The count is initialized to 1 and the test expression is evaluated. Since, the test expression count <= n (1 less than or equal to 10) is true, the body of for loop is executed and the value of sum will be equal to 1.
Then, the update statement ++count is executed and count will be equal to 2. Again, the test expression is evaluated. The test expression is evaluated to true and the body of for loop is executed and the sum will be equal to 3. And, this process goes on. Eventually, the count is increased to 11. When the count is 11, the test expression is evaluated to 0 (false) and the loop terminates.

## continue statement in C

Continue statement:
C – continue statement in C with example. Continue statement is mostly used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop's body for the current iteration.

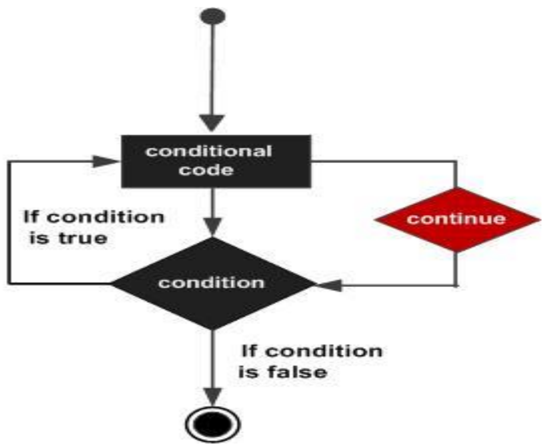
The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops,**continue** statement causes the program control to pass to the conditional tests.

Syntax:

The syntax for a **continue** statement in C is as follows −

```
continue;
```

Flow Diagram:



Example

```
#include <stdio.h>
int main () {

   /* local variable definition */
   int a = 10;
   /* do loop execution */
   do {
      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         continue;
      }
      printf("value of a: %d\n", a);
      a++;
   } 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: 16
value of a: 17
value of a: 18
value of a: 19
```