

For loop in C

A loop is used for executing a block of statements repeatedly until a particular condition is satisfied. For example, when you are displaying number from 1 to 100 you may want set the value of a variable to 1 and display it 100 times, increasing its value by 1 on each loop iteration.

In C, we have three types of basic loops: **for**, **while** and **do-while**.

Syntax of for loop

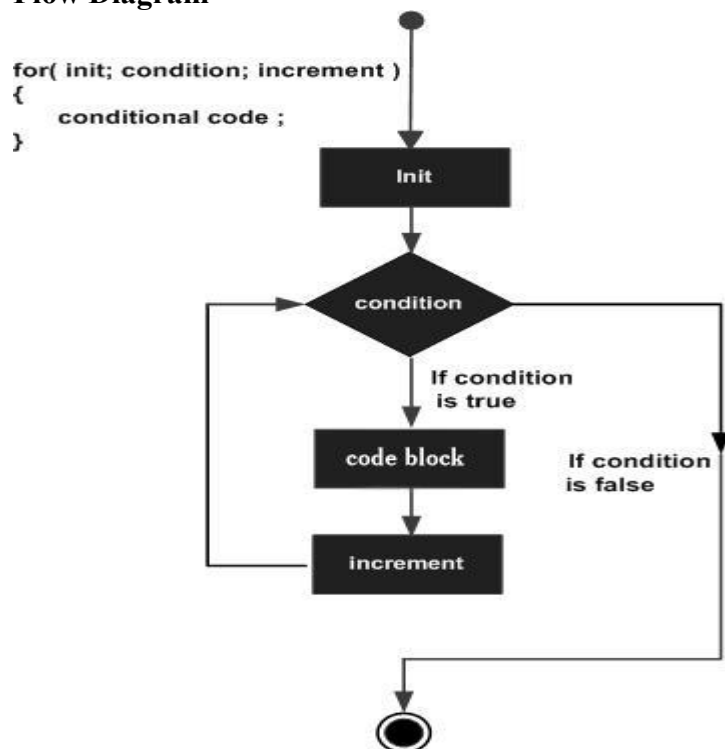
```
for(initialization; condition ; increment/decrement){  
    C statement(s);  
}
```

1 to 5	10 to 1	15 to 5
<code>for(int i= 1; i<=5; ++i){</code>	<code>for(int i=10; i>=1; --i)</code>	<code>for(int i=15; i>4; --i)</code>
<code> printf("%d", i);</code>		
<code>}</code>		

Flow of Execution of the for Loop

As a program executes, the interpreter always keeps track of which statement is about to be executed. We call this the control flow, or the flow of execution of the program.

Flow Diagram



First step: In for loop, initialization happens first and only once, which means that the initialization part of for loop only executes once.

Second step: Condition in for loop is evaluated on each loop iteration, if the condition is true then the statements inside for loop body gets executed. Once the

condition returns false, the statements in for loop does not execute and the control gets transferred to the next statement in the program after for loop.

Third step: After every execution of for loop's body, the increment/decrement part of for loop executes that updates the loop counter.

Fourth step: After third step, the control jumps to second step and condition is re-evaluated.

The steps from second to fourth repeats until the loop condition returns false.

Example of a Simple For loop in C with continue

Here in the loop initialization part I have set the value of variable i to 1, condition is $i \leq 6$ and on each loop iteration the value of i increments by 1.

```
#include <stdio.h>
int main(){
    for(int i=1; i<=6; i++){
        if(i== 4 )
            continue;
        printf("Value of variable i is:%d", i);
    }
    return 0;}
```

Output:

Value of variable i is: 1Value of variable i is: 2Value of variable i is: 3Value of variable i is: 4Value of variable i is: 5Value of variable i is: 6

Example of a Simple For loop in C with break

```
#include <stdio.h>
int main(){
    for(int i=1; i<=6; i++){
        if(i== 4 )
            break;
        printf("Value of variable i is:%d", i);
    }
    return 0;}
```

Output:

Value of variable i is: 1Value of variable i is: 2Value of variable i is: 3Value of variable i is: 4

Infinite for loop in C

A loop is said to be infinite when it executes repeatedly and never stops. This usually happens by mistake. When you set the condition in for loop in such a way that it never return false, it becomes infinite loop.

For example:

```
#include <stdio.h>
```

```
int main(){
    for(int i=1; i>=1; i++){
        printf("Value of variable i is: %d \n",i);
    }
    return 0;}
```

This is an infinite loop as we are incrementing the value of i so it would always satisfy the condition $i \geq 1$, the condition would never return false.

Here is another example of infinite for loop:

```
// infinite loop for ( ; ; ) {
// statement(s)}
```

Example: for loop

Q1. 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, i, sum = 0;

    printf("Enter a positive integer: "); //10
    scanf("%d", &num);

    for(i = 1; i <= num; ++i)
    {
        sum += i;
    }

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

    return 0;
}
```

Output

Enter a positive integer: 10

Sum = 55

Q2. Factorial of a given number

```
#include <stdio.h>
int main()
{
    int n, i;
    unsigned long long factorial = 1;

    printf("Enter an integer: ");
    scanf("%d",&n);
```

```

// show error if the user enters a negative integer
if(n < 0)
    printf("Error! Factorial of a negative number doesn't exist.");

else
{
    for(i=1; i<=100; ++i)
    {
        factorial *= i;          // factorial = factorial*i;
    }
    printf("Factorial of %d = %llu", n, factorial);
}

return 0;
}

```

while Loop

As discussed earlier, loops are used for executing a block of program statements repeatedly until the given loop condition returns false.

Syntax of while loop

```

while(condition){
    statement(s);
}

```

How while Loop works?

In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute, this happens repeatedly until the condition returns false. When condition returns false, the control comes out of loop and jumps to the next statement in the program after while loop.

Note: The important point to note when using while loop is that we need to use increment or decrement statement inside while loop so that the loop variable gets changed on each iteration, and at some point condition returns false. This way we can end the execution of while loop otherwise the loop would execute indefinitely.

```

int i=10; // initialization
while(i>5){ // condition
    printf("%d", i); // statement
    i--; // update
}

```

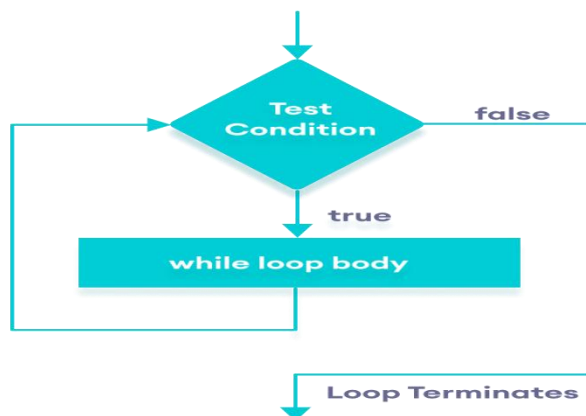
Output:

```

10
9
8
7
6

```

Flowchart of while Loop



While Loop example in C

```
#include <stdio.h>
int main(){
    int i=1;
    while(i<=6){
        printf("Value of variable i is: %d",i);
        i++; // updating
    }
}
```

Output:

Value of variable i is: 1 Value of variable i is: 2 Value of variable i is: 3 Value of variable i is: 4 Value of variable i is: 5 Value of variable i is: 6

```
#include <stdio.h>
int main(){
    int i=1;
    int j= 6;
    while(i<=6 && j >=4){
        printf("Value of variable i is:%d \t Value of variable j is: %d", i, j);
        i++; j--;
    }
}
```

Output ??

Infinite While loop

A while loop that never stops is said to be the infinite while loop, when we give the condition in such a way so that it never returns false, then the loops becomes infinite

and repeats itself indefinitely.

An example of infinite while loop:

This loop would never end as I'm decrementing the value of i which is 1 so the condition $i \leq 6$ would never return false.

```
#include <stdio.h>
int main() {
    int i=1;
    while(i<=6) {
        printf("Value of variable i is: %d", i);
        i--;
    }
}
```

Do while Loop

Do-while loop is similar to while loop, however there is a difference between them: In while loop, condition is evaluated first and then the statements inside loop body gets executed, on the other hand in do-while loop, statements inside do-while gets executed first and then the condition is evaluated.

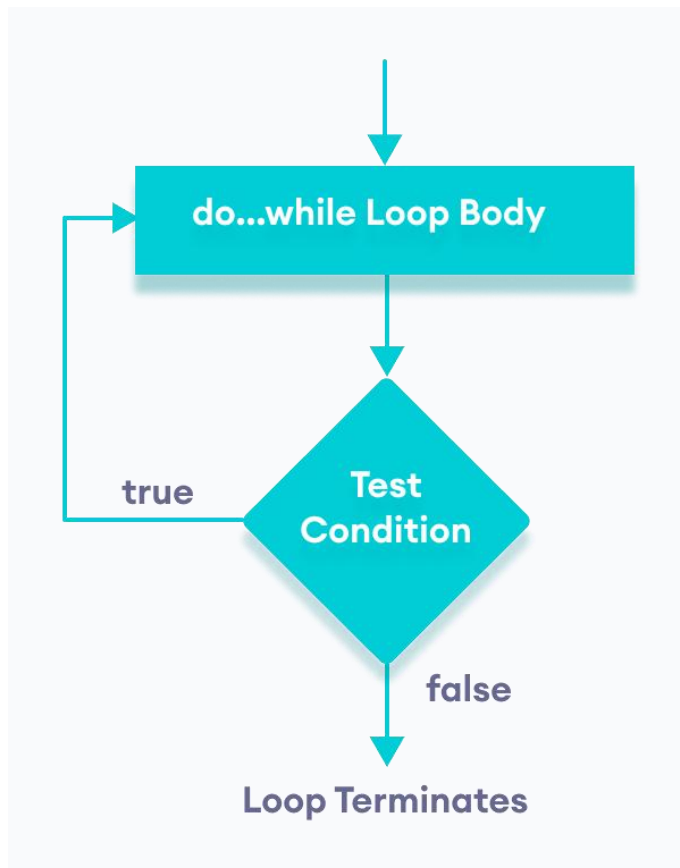
Syntax of do-while loop

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

How do-while loop works?

First, the statements inside loop execute and then the condition gets evaluated, if the condition returns true then the control jumps to the “do” for further repeated execution of it, this happens repeatedly until the condition returns false. Once condition returns false control jumps to the next statement in the program after do-while.

Flowchart of do...while Loop



do-while loop example in C

```
#include <stdio.h>
int main(){
    int num=1;
    do{
        printf("Value of num: %d ", num);
        num++;
    }while(num<=6);
    return 0;}
```

Output:

Value of num: 1Value of num: 2Value of num: 3Value of num: 4Value of num: 5Value of num: 6

Nested Loop-

A nested loop is a loop in which one loop resides inside another loop where the inner loop gets executed first, satisfying all the set of conditions that prevailed within the loop followed by an outer loop set of conditions. Execution of statements within the loop flows in a way that the inner loop of the nested loop gets declared, initialized and

then incremented. Once all the condition within the inner loop gets satisfied and becomes true, it moves for the search of the outer loop. It is often called a “loop within a loop”.

Syntax of Nested Loop in C

Following are the syntax:

1. Syntax of Nested for Loop

```
for (initialization; condition; increment) {  
    for (initialization; condition; increment) {  
        // set of statements for inner loop  
    }  
    //set of statement for outer loop  
}
```

2. Syntax of Nested While loop

```
while(condition) {  
    while(condition) {  
        // set of statement of inside while loop  
    }  
    //set of statement for outer while loop  
}
```

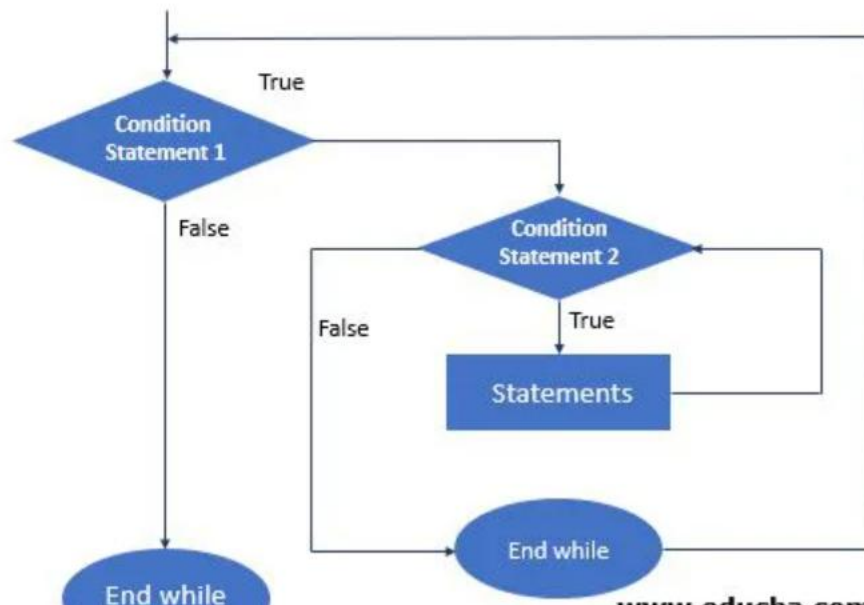
3. Syntax of Nested Do-While loop

```
do {  
    while(condition) {  
        for (initialization; condition; increment) {  
            //set of statement of inside do-while loop  
        }  
        // set of statement of inside do-while loop  
    }  
    //set of statement of outer do-while loop  
} while(condition);
```

Explanation:

In any loop, the first execution will happen for the set of statements of the inner loop. If the condition gets satisfied and is true, it will again come inside and satisfy the second condition. If that gets false, it will search for the outer loop and try to satisfy all the conditions. In this way, the flow continues. Also, it is not mandatory that the loop must be placed within the loop of the same genre. It can be any loop which means any loop can be placed inside any loop.

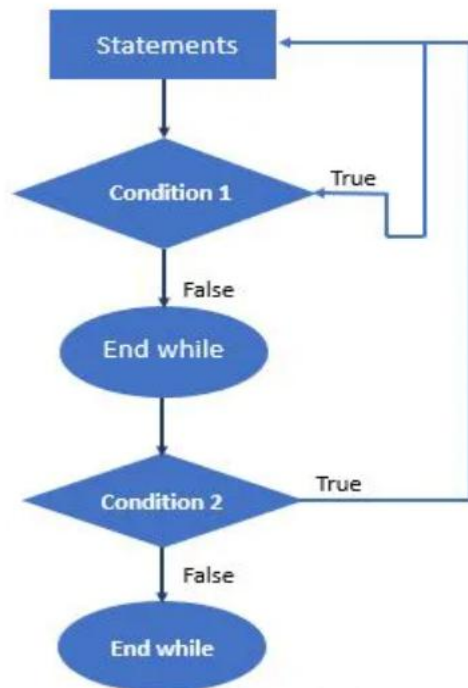
1. Flowchart of Nested While Loop



Explanation:

Initially, one condition statement is being provided in the while loop; if that condition of inner loop condition statement is true, then loop execution will continue with the same inner loop condition forming a loop as soon as it finds that the condition is false it will come out of the inner while loop and search for the outer loop condition. If the outer loop condition comes out to be true, then it will execute all the sets of statements and information. After following all set of instructions, it will become false, and the loop will come out to the main program control saying end while loop.

2. Nested Do While Loop



Explanation:

A statement is provided as an input followed by a condition which checks whether it satisfies the condition or not; if it satisfies the condition, then again looping will happen; if not, it will come out of the loop and will check for a false condition then while loop gets end then again one more loop gets into execution which will also follow the similar pattern of execution.

3. For Loop

Explanation:

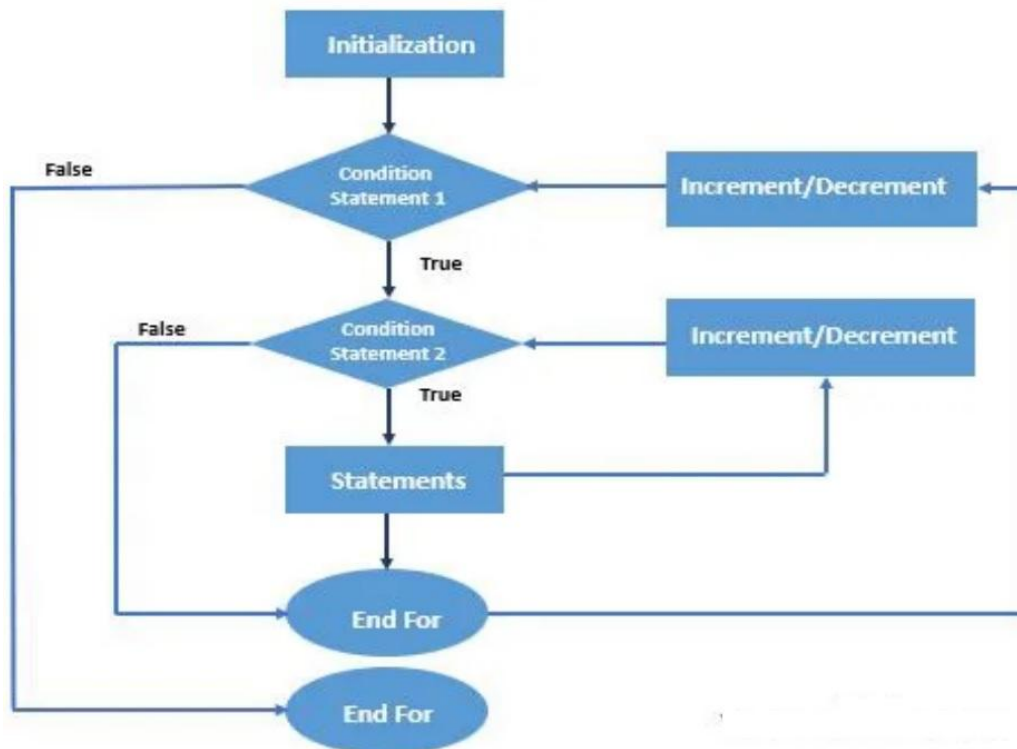
A for loop includes the initialization of variable followed by condition 1; if condition 1 gets satisfied, it will search for the second condition if that also gets true, it will get incremented and then when the condition satisfies, and it gets true, then it searches for the new set of statements, following the same flow of execution. Once the condition becomes false, it comes out of the loop and searches for an end of for statement, reaching back to the entire program execution's main condition.

How Nested Loop works in C ?

Looping plays a very pivotal role in any programming language; the same it happens in the case of C. When one loop resides inside another loop is called nesting. When we loop two loops together, i.e. kind of nesting, then the outer loop takes control of the number of times the inner loop works and takes care of all manipulation and computation.

Examples of Nested Loop in C

Given below are the examples of Nested Loop in C:



```
#include <stdio.h>
int main() {
    for (int i = 1; i <= 5; ++i) {
        for (int j = 1; j <= 3; ++j) {
            printf( "* ");
        }
        printf("\n ");
    }
    return 0;
}
```

```
#include <stdio.h>
```

```

    int main() {
    int rows, i = 1;
    printf( "Enter the number of rows: ");
    scanf("%d",& rows);
    while (i <= rows) {
        int j = 1;
        while(j <= i) {
            printf("%d", i);
            j++;
        }
        printf( "\n");
        i++;
    }
    return 0;
}

```

```

#include <stdio.h>
int main() {
    for (int i = 1; i <= 5; ++i) {
        for (int j = 1; j <= 3; ++j) {
            printf("%d  %d", i, j) ;
        }
        printf("\n");
    }
    return 0;
}

```

Example #1

Nested loop with a while loop for getting all the numbers from 1 – 20.

Code:

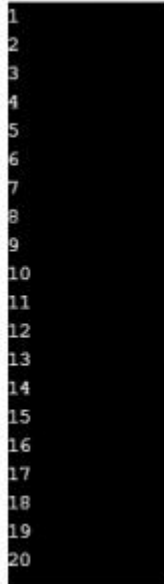
```

#include <stdio.h>
int main () {
    int a = 1;
    while (a <= 20) {
        printf("\n %d", a);
        a++;
    }
}

```

```
return 0;
}
```

Output:



```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Example #2

Program with a while nested loop to verify whether the number is odd or even.

Code:

```
#include <stdio.h>
int main () {
int choose = 1;
while (choose == 1) {
int b;
printf( "Enter a number to verify odd or even\n");
scanf("%d", &b);
if (b%2 == 0) {
printf( " Number is even\n");
}
else {
printf( " Number is odd \n");
}
printf("To check for more: 1 for yes and 0 for no \n");
scanf("%d", &choose);
}
printf( "All numbers are verified \n");
return 0;
}
```

Output:

```
Enter a number to verify odd or even
56
    Number is even
To check for more: 1 for yes and 0 for no
1
Enter a number to verify odd or even
87
    Number is odd
To check for more: 1 for yes and 0 for no
0
All numbers are verified
```

Example #3

Program with a nested for loop to calculate the sum of given numbers within the specified range.

Code:

```
#include <stdio.h>
int main () {
    int sum = 0, c, d;
    for (c = 0; c < 10; c++) {
        printf("Enter the number\n");
        scanf("%d", &d);
        sum = sum + d;
    }
    printf("Sum is %d ", sum);
    return 0;
}
```

Output:

```
Enter the number
3
Enter the number
6
Enter the number
534
Enter the number
85
Enter the number
23
Enter the number
74
Enter the number
62
Enter the number
85
Enter the number
13
Enter the number
98
Sum is 983
```

Example #4

Program to print the numbers using a do-while loop.

Code:

```
#include<stdio.h>
int main () {
int k = 1;
do {
printf("%d", k);
k++;
} while (k <= 15);
return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

Example #5

Program to get the multiplication table of a given number in the m*n format using nested for loop.

Code:

```
#include <stdio.h>
int main () {
int m;
int n;
for (m = 10; m <= 11; m++) {
printf( "Table of  %d ", m);
for (n = 1; n <= 10; n++) {
printf(“ %d * %d = %d” ,m, n , m*n);
}
}
return 0;
}
```

Output:

Table of 10

$$10 \times 1 = 10$$

$$10 \times 2 = 20$$

$$10 \times 3 = 30$$

$$10 \times 4 = 40$$

$$10 \times 5 = 50$$

$$10 \times 6 = 60$$

$$10 \times 7 = 70$$

$$10 \times 8 = 80$$

$$10 \times 9 = 90$$

$$10 \times 10 = 100$$

Table of 11

$$11 \times 1 = 11$$

$$11 \times 2 = 22$$

$$11 \times 3 = 33$$

$$11 \times 4 = 44$$

$$11 \times 5 = 55$$

$$11 \times 6 = 66$$

$$11 \times 7 = 77$$

$$11 \times 8 = 88$$

$$11 \times 9 = 99$$

$$11 \times 10 = 110$$