

PF Lab # 5



September 18, 2019

Nida Munawar

Romasha Khursheed

**Control statements**

In looping, a program executes the sequence of statements many times until the stated condition becomes false. A loop consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.

# Types of Loops

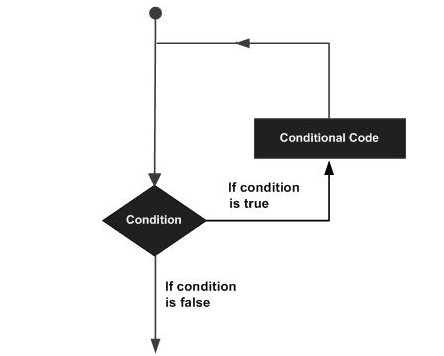
Depending upon the position of a control statement in a program, a loop is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.



The loop that does not stop executing and processes the statements number of times is called as an **infinite loop**. An infinite loop is also called as an "**Endless loop**."

1. No termination condition is specified.
2. The specified conditions never meet.

'C' programming language provides us with three types of loop constructs:

1. The while loop

2. The do-while loop

3. The for loop

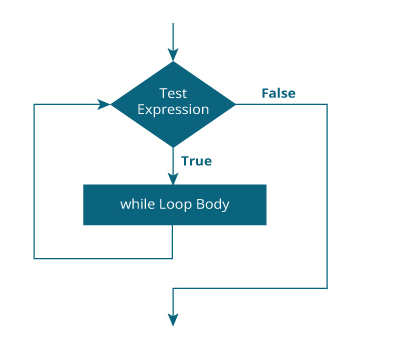
## While Loop

The basic format of while loop is as follows:

while (condition) {

statements;

}



It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.

After exiting the loop, the control goes to the statements which are immediately after the loop.

In while loop, if the condition is not true, then the body of a loop will not be executed, not even once.

#include<stdio.h>

#include<conio.h>

int main()

{

int num=1; //initializing the variable

while(num<=10) //while loop with condition

{

printf("%d\n",num);

num++; //incrementing operation

}

return 0;

}

## Do-While loop

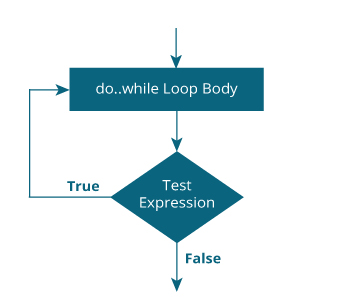
A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of while loop is as follows:

do {

statements

} while (expression);



As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed.

The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

#include<stdio.h>

#include<conio.h>

int main()

{

int num=1; //initializing the variable

do //do-while loop

{

printf("%d\n",2\*num);

num++; //incrementing operation

}while(num<=10);

return 0;

}

## For loop

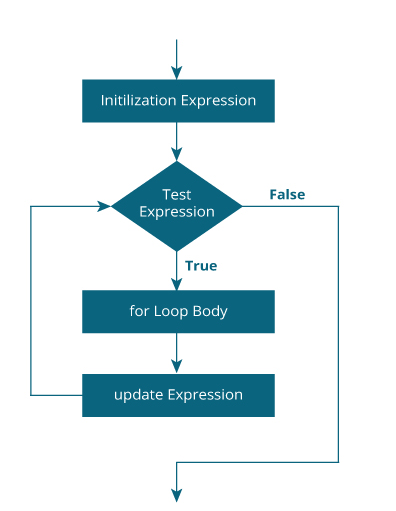
A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

for (initial value; condition; incrementation or decrementation )

{

statements;

}



* The initial value of the for loop is performed only once.
* The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
* The incrementation/decrementation increases (or decreases) the counter by a set value.

Following program illustrates the use of a simple for loop:

#include<stdio.h>

int main()

{

int number;

for(number=1;number<=10;number++) //for loop to print 1-10 numbers

{

printf("%d\n",number); //to print the number

}

return 0;

}

# Nested loop in C

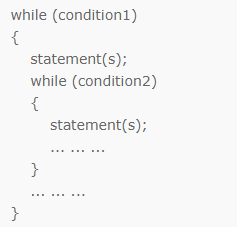
A loop inside another loop is called a nested loop. The depth of nested loop depends on the complexity of a problem. We can have any number of nested loops as required. Consider a nested loop where the outer loop runs n times and consists of another loop inside it. The inner loop runs m times. Then, the total number of times the inner loop runs during the program execution is n\*m.

## Types of nested loops

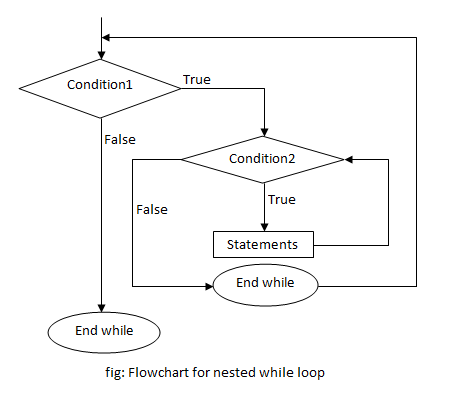
* [Nested while loop](https://www.programtopia.net/c-programming/docs/nested-loop#while)
* [Nested do-while loop](https://www.programtopia.net/c-programming/docs/nested-loop#do-while)
* [Nested for loop](https://www.programtopia.net/c-programming/docs/nested-loop#for)

**Note**: There can be mixed type of nested loop i.e. a for loop inside a while loop, or a while loop inside a do-while loop.

### Syntax of Nested while loop

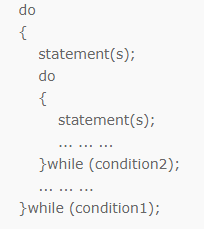


### Flowchart of Nested while loop



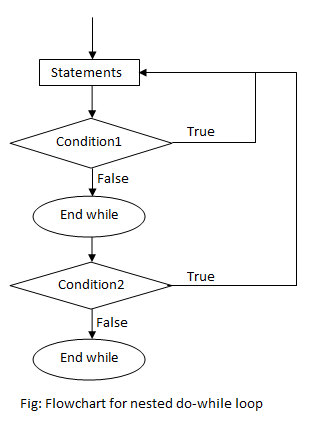
## Nested do-while loop

A do-while loop inside another do-while loop is called nested do-while loop.



### Syntax of Nested do-while loop

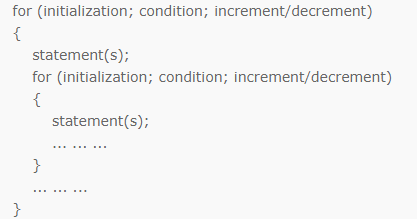
### Flowchart of Nested do-while loop



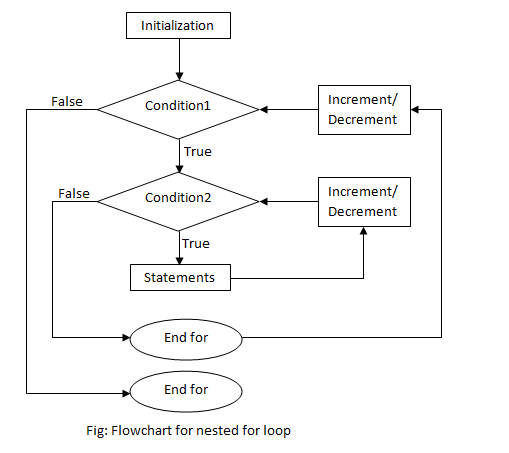
## Nested for loop

A for loop inside another for loop is called nested for loop.

### Syntax of Nested for loop



### Flowchart of Nested for loop



#include <stdio.h>

int main() {

int i, j;

int table = 2;

int max = 5;

for (i = 1; i <= table; i++) { // outer loop

for (j = 0; j <= max; j++) { // inner loop

printf("%d x %d = %d\n", i, j, i\*j);

}

printf("\n"); /\* blank line between tables \*/

}}

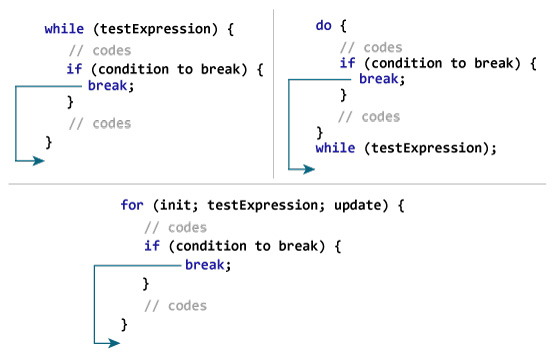
# C break

The break statement ends the loop immediately when it is encountered. Its syntax is:

1. break;

The break statement is almost always used with if...else statement inside the loop.

### How break statement works?



# C continue

The continue statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

1. continue;

The continue statement is almost always used with the if...else statement.

### How continue statement works?

