Loop Statements. Pre-condition and Post-condition Loops

Looping

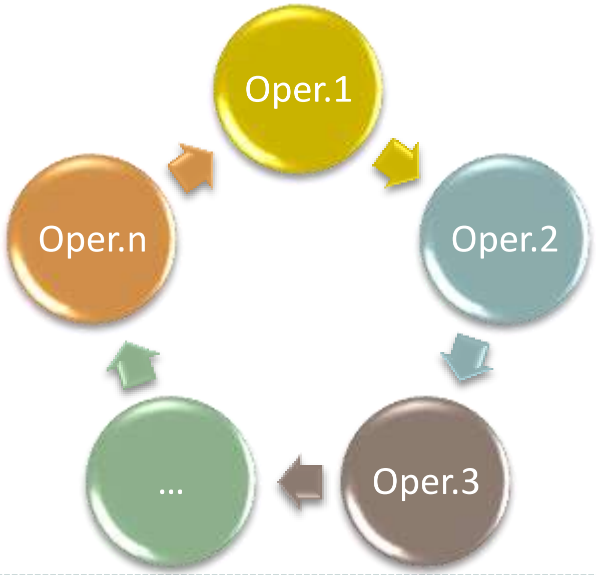
One of the most common programming tasks is to perform the same set of statements multiple times. Rather than repeat a set of statements again and again, a loop can be used to perform the same set of statements repeatedly.

A ***Loop Statement*** is a series of instructions that is repeated until a certain condition is met.

Looping

Each pass through the loop is called an ***iteration***.

A loop can be viewed as a cycle repeating a specific number of operations.



Types of looping

Depending on the position of the condition in the loop there are two types of loops:

- Pre-condition (pre-test) loop
- Post-condition (post-test) loop

The difference between Pre-test and Post-test Loops

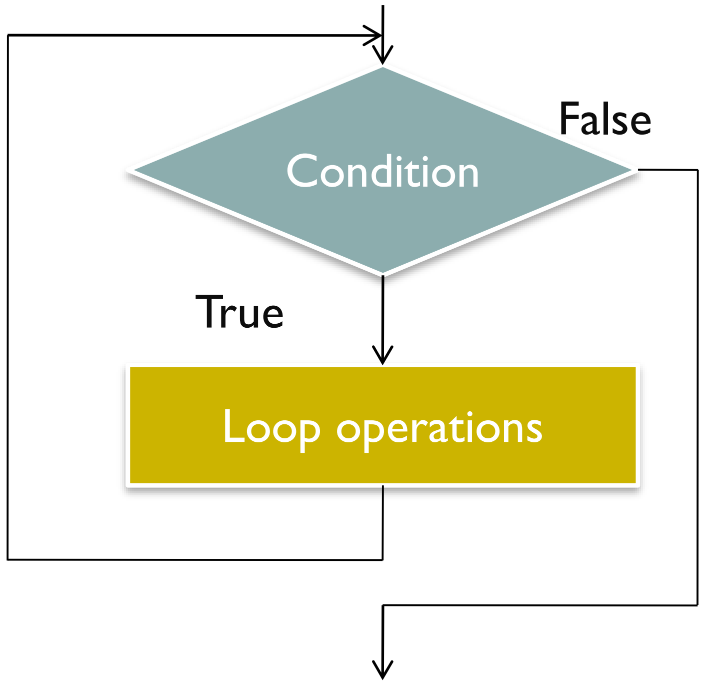
A Pre-test Loop is one in which the block of code, named the body of the loop, is to be repeated while the specified condition is true, and the condition is situated at the beginning of the loop and is tested before the loop is executed.

A Post-test Loop is one in which the block of code is to be repeated while the specified condition is true, and the condition is situated at the end of the loop and is tested after the loop is executed.

The Pre-test Loop

The key feature of the pre-test loop is that we test to see whether or not to continue before executing the body of the loop. The effect of this is that we might not execute the body at all. So, if we're designing a program that has a section which might be executed 0 or more times, a pre-test structure is what we want.

The flowchart for the pre-test loop structure:

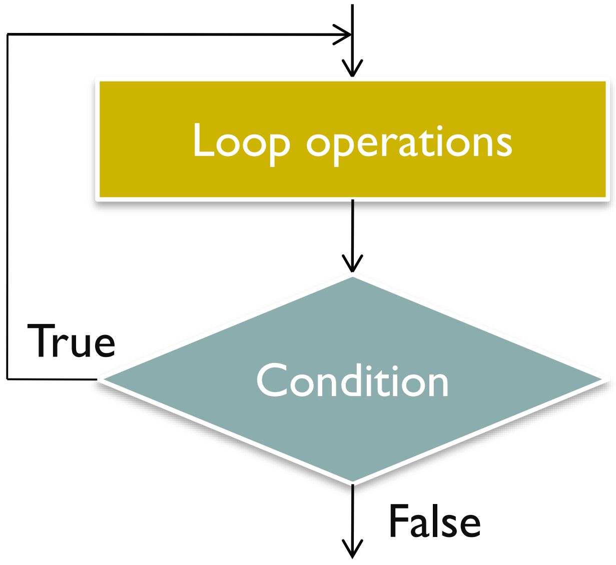


The Post-test Loop

In the post-test loop the condition is at the end of the loop. This means that this loop will always be executed at least once, before the test is made to determine whether it should continue.

This is the only difference between pre-test and post-test loops.

The flowchart for the post-test loop structure:



Loop Statements in C

C language gives us a choice of three types of loop statements:

- While
- For
- Do - while

The ***while*** and ***for*** loop statements refer to ***Pre-test loop statements.***

The ***do – while*** loop statement corresponds to ***Post-test loop statements.***

While and For Loop Statements

While Loop Statement

The ***while*** loop statement structure consists of a block of code and a condition. The loop will repeat as long as the condition is true. The while tests its condition at the top of the loop. Therefore, if the condition is false to begin with, the loop will not execute at all.

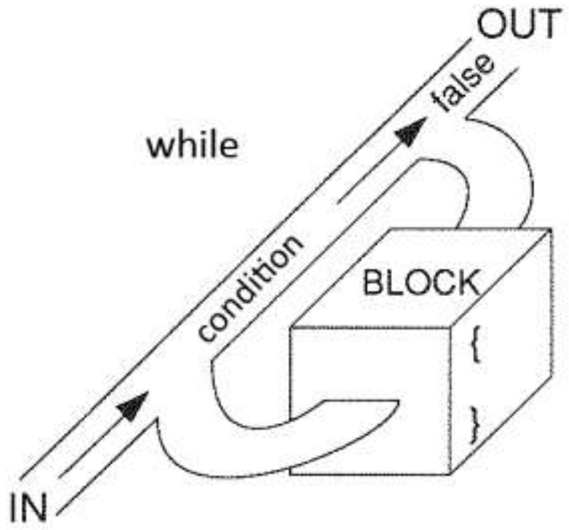
Because the condition of the loop is tested before the block is executed, the while loop is called a ***pre-test loop statement.***

While Loop Statement

The syntax of the *__while__* loop statement:

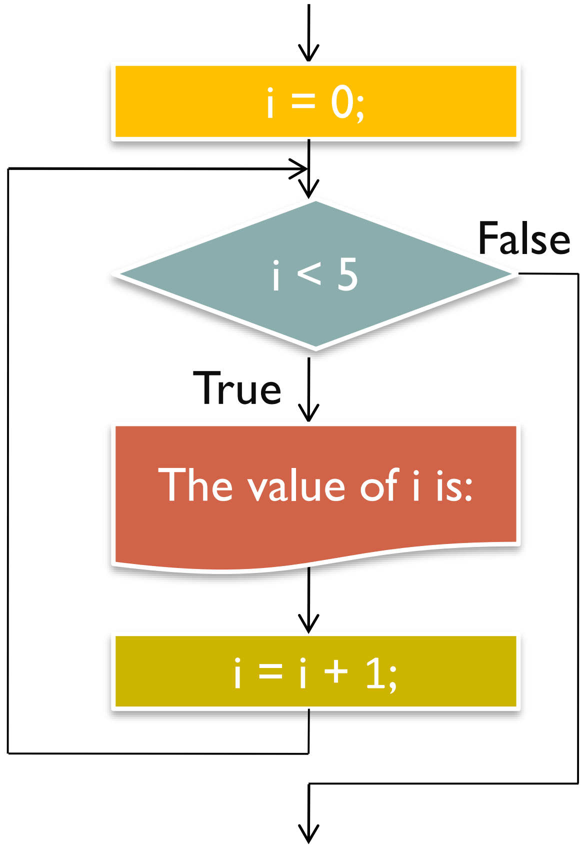
```
  initialization;
  while(condition)
 {
  block of
statements (body of
the loop);
  }
```

While loop can be graphically viewed as follows:



While Loop Statement

The flowchart of the ***while*** loop statement:



Example of program using ***while*** loop statement:

```
 #include <stdio.h>
 int main()
 {
  int i = 0;
  while (i<5)
    {
      printf("The value
of i is %d\n", i);
      i = i + 1;
     }
   return o;
  }
```

For Loop Statement

The ***for*** loop is usually used when the number of iterations is predetermined. Similar to the while loop, the for loop is a ***pre-test loop.***

Unlike other kinds of loops, such as the while loop, the for loop is often distinguished by an explicit loop counter or loop variable. This allows the body of the for loop (the code that is being repeatedly executed) to know about the sequencing of each iteration.

For Loop Statement

The syntax for a 'for loop' is the following:

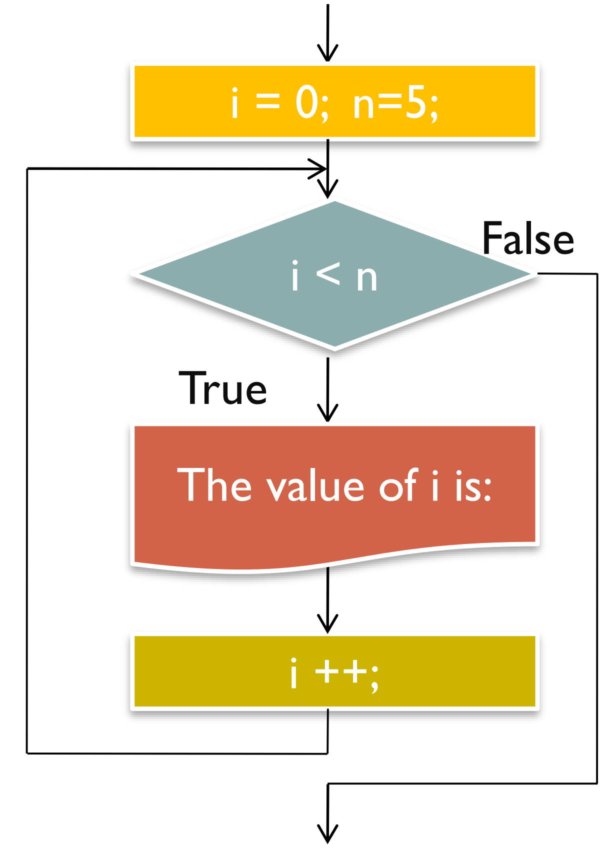
```
for(initialization; condition; update)
{
 block of statements (body of the loop);
}
```

General features of a for loop statement:

- `<initialization>` sets the initial value of the loop control variable, it is executed only once before looping
- `<condition>` is a Boolean expression, it tests the value of the loop control variable. If not given, it is assumed to be true. If `<condition>` is false, the loop is terminated
- `<update>` updates the loop control variable

For Loop Statement

The flowchart of the ***for*** loop statement: (similar to while)

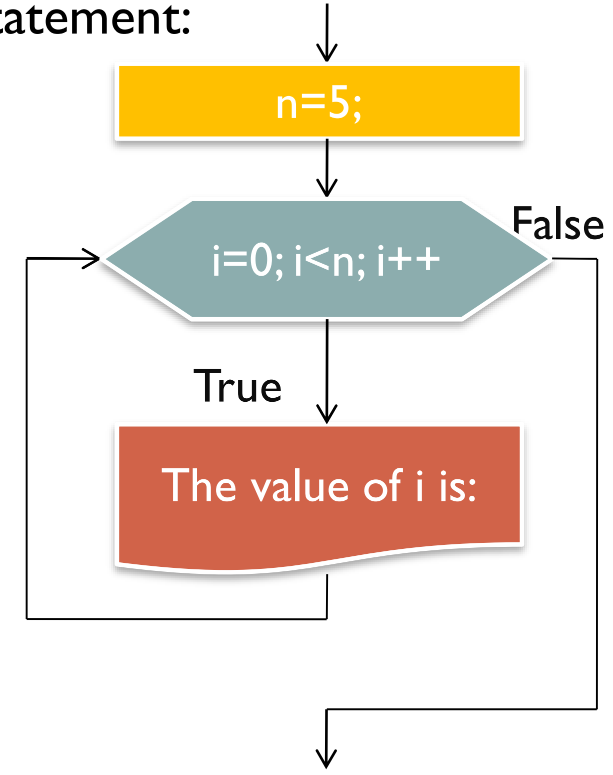


Example of program using ***for*** loop statement:

```
#include <stdio.h>
int main()
  {
   int i, n = 5;
   for(i=0; i<n; i++)
         {
     printf(“The value
       of i is %d \n”,i);
   }
    return 0;
  }
```

For Loop Statement

There is another version of flowchart for '***for*** loop' statement:



Advantage of for loop:

Any for loop is equivalent to some while loop, so the language doesn't get any additional power by having the for statement.

For certain type of problem, a for loop can be easier to construct and easier to read than the corresponding while loop.

The for statement makes a common type of while loop easier to write. It is a very good (perhaps the best) choice for counting loops.

Do - While Loop Statement

Do-While Loop Statement

The ***do - while*** construct consists of a block of code and a condition. First, the code within the block is executed, and then the condition is evaluated. If the condition is true the code within the block is executed again. This repeats until the condition becomes false.

Because do while loops check the condition after the block is executed, the control structure is often also known as a **post-test loop**.

The do-while loop statement is used when you want to execute the loop body at least once.

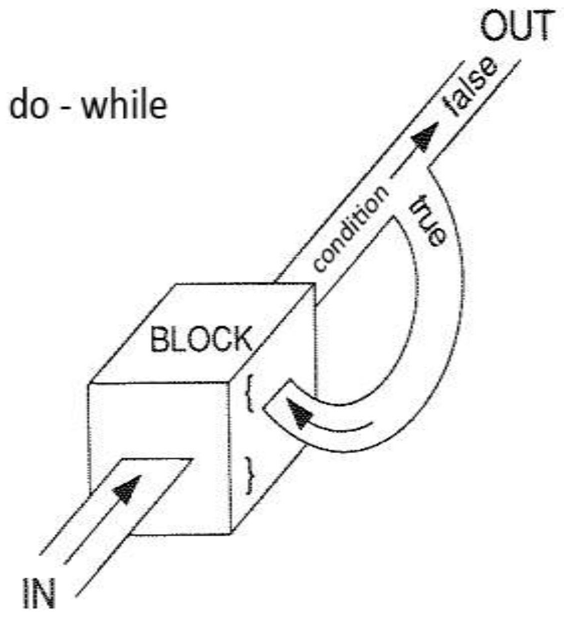
Do-While Loop Statement

The syntax of the ***do-while*** loop statement:

```
<initialization>
do
{
   <statement block>;
   <update>;
} while(<condition>);
```

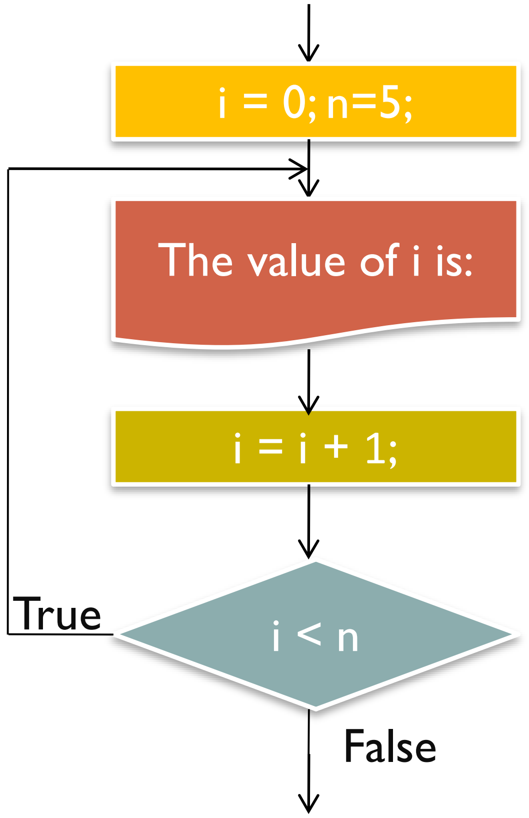
!!! Note that a semi-colon (;) must be used at the end of the do-while loop !!!

Do-While loop can be graphically viewed as follows:



Do-While Loop Statement

The flowchart of the ***do-while*** loop statement:



Example of program using ***do-while*** loop statement:

```
 #include <stdio.h>
 int main()
{
int i,n = 5;
i = 0;
do
    {
  printf("The value
     of i is %d \n",i);
  i = i +1;
     } while( i<n) ;
return 0;
}
```

Conclusion

- Instead of performing the same set of statements multiple times use a loop statement to perform the same set of statements repeatedly;
- C language gives you a choice of three loop statements, while, do while and for;
- The ***while loop*** keeps repeating an action until an associated test returns false. This is useful where the programmer does not know in advance how many times the loop will be traversed;
- The ***do while loop*** is similar, but the test occurs after the loop body is executed. This ensures that the loop body is run at least once;
- The ***for loop*** is frequently used, usually where the loop will be traversed a fixed number of times;