Looping in C Programming

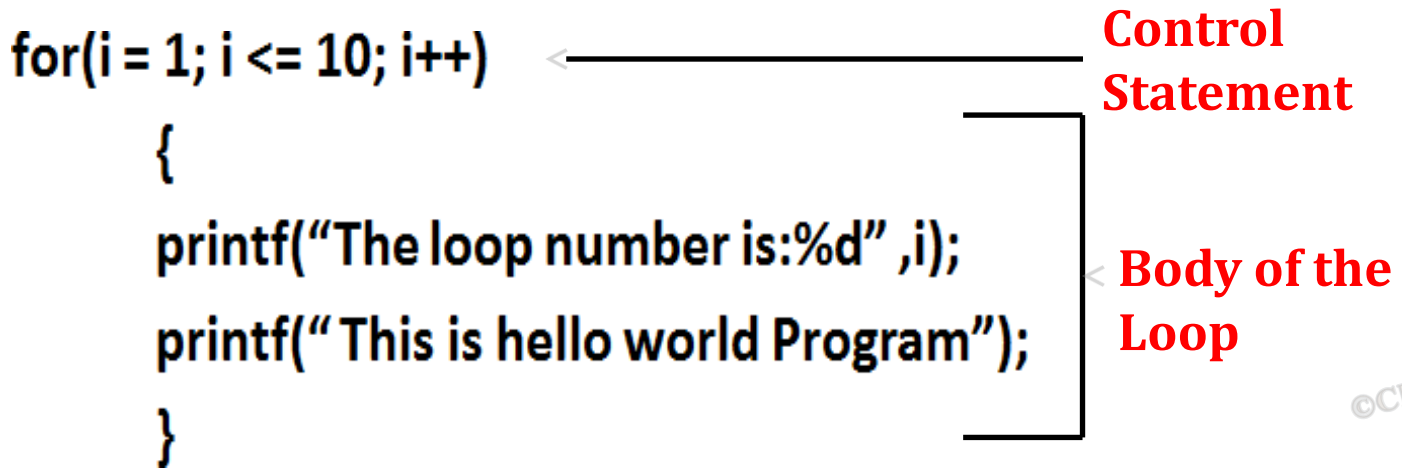
Dr. Chayan Halder
Assistant Professor
Ramakrishna Mission Vivekananda Centenary College, Kolkata

Introduction

- One of the fundamental properties of a programming language is the ability to repetitively execute a sequence of statements which is called decision making and looping. "**Decision making and looping**" is one of the most important concepts of computer programming.
- Programs should be able to make decisions based on the condition provided and then repeat a specific chunk of code or statement again and again.
- These looping capabilities enable programmers to develop concise programs containing repetitive processes that could otherwise require an excessive number of statements. It enable us to repeat a specific section of code or statement without the use of goto statements .
- In looping a sequence of statements are executed until some conditions for termination of loop are satisfied.

Loop Segments

- A program loop therefore consists of two segments:
- One known as the **body of the loop** and the 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.



Looping Types

- Depending on the position of the control statement in the loop, a loop control structure may be classified either as

1. The entry controlled loop
2. The exit controlled 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

1. The counter controlled loop
2. The sentinel controlled loop

Entry Controlled Loop

- The types of loop where the test condition is stated before the body of the loop, are known as the entry controlled loop. So in the case of an entry controlled loop, the condition is tested before the execution of the loop.
- If the test condition is true, then the loop gets the execution, otherwise not. For example, the **for loop** is an entry controlled loop. In the given figure, the structure of an entry controlled loop is shown.

Entry Controlled Loop- Flowchart

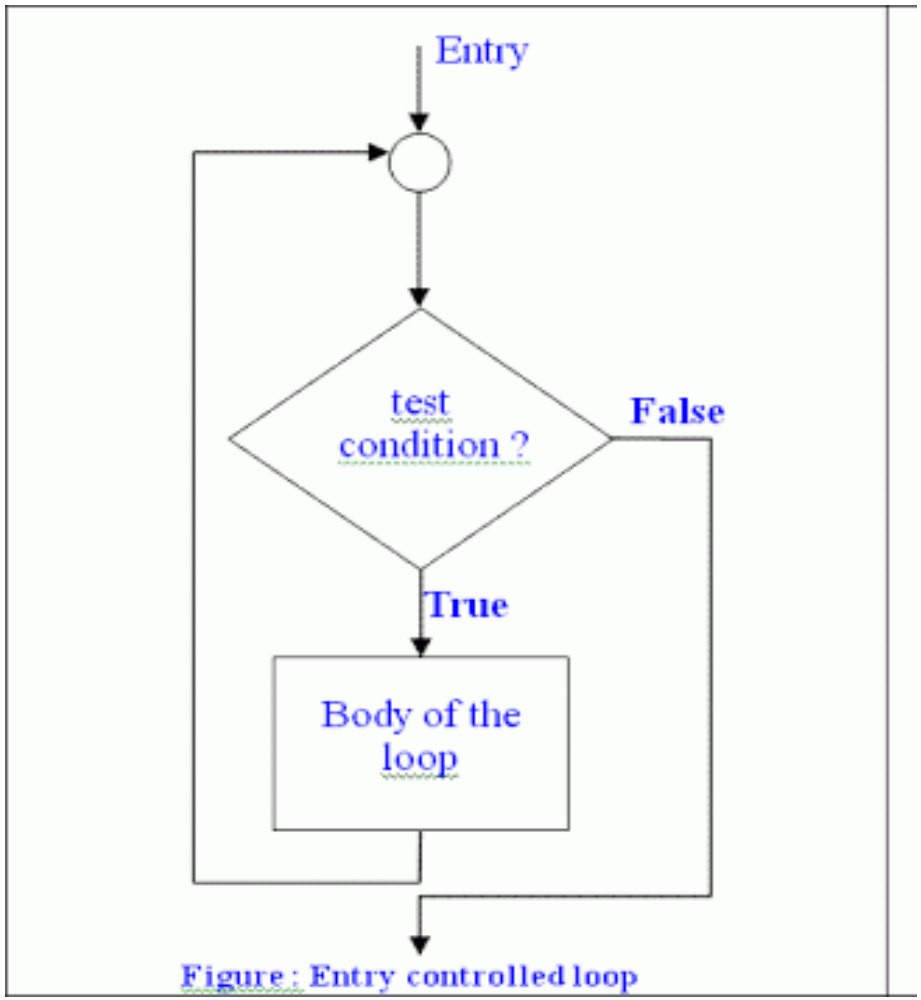


Figure : Entry controlled loop

Exit Controlled Loop

- The types of loop where the test condition is stated at the end of the body of the loop, are know as the exit controlled loops. So, in the case of the exit controlled loops, the body of the loop gets execution without testing the given condition for the first time. Then the condition is tested.
- If it comes true, then the loop gets another execution and continues till the result of the test condition is not false.
- In case of exit controlled loop the body of the loop is executed unconditionally for the first time without testing the condition.
- For example, the do statement or the do....while loop is an exit controlled loop. The structure of an exit controlled loop is given in the given figure.

Exit Controlled Loop- Flowchart

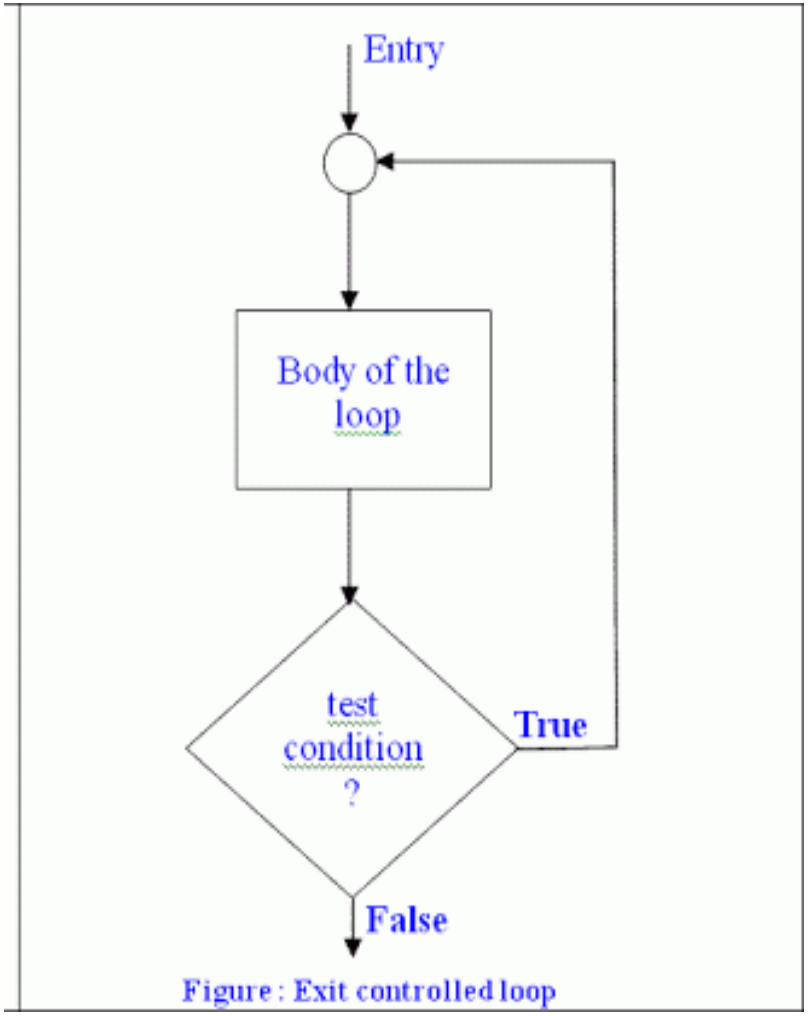


Figure : Exit controlled loop

Entry VS Exit Controlled Loop

- **Test condition:** Test condition appears at the beginning in entry controlled loop where Test condition appears at the end in exit controlled loop.
- **Control variable:** In entry controlled loop control variable is counter variable but in exit controlled loop Control variable can be counter & sentinel variable
- **Execution:** In entry controlled loop each execution occurs by testing condition but in exit controlled loop each execution except the first one occurs by testing condition

Counter Controlled Loop

- 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. The counter must be initialized, tested and updated properly for the desired loop operations. The number of times we want to execute the loop may be a constant or variable.
- A counter controlled loop is also called definite repetition loop because the number of repetitions is known before the loop begins executing.
- **Example :** A while loop is an example of counter controlled loop.

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.
- A sentinel controlled loop is sometimes called indefinite repetition loop because the number of repetitions is not known before the loop begins executing.
- **Example :** The following **do....while** loop is an example of sentinel controlled loop.

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

Counter VS Sentinel Loop

- **Number of execution:** In counter controlled loop execution occurs according to the previously known number but in sentinel controlled loop unknown number of execution occurs
- **Condition variable:** In counter controlled loop condition variable is known as counter variable but in sentinel controlled loop condition variable is known as sentinel variable
- **Value and limitation of variable:** In counter controlled loop the value of the variable and the limitation of the condition for the variable both are strict but in sentinel controlled loop the limitation for the condition variable is strict but the value of the variable varies in this case

Steps of Looping a Process

- A looping process, in general, would include the following four steps. The given steps may come at any order depending on the type of the loop.

1. Setting and initialization of a condition variable.
2. Execution of the statements of the loop.
3. Test for a specified value of the condition variable for execution of the loop.
4. Incrementing or updating the condition variable.

Types of Loops in C

- The C language supports three types of looping statement:

1. The for statement
2. The do while statement
3. The while do statement

The for Statement

- The for loop is an entry controlled loop. It has the following structure:

```
for(initialization; test-condition; increment/decrement)
    {
        body of the loop;
    }
```

Analyze the for Statement

for → The keyword of for loop

Initialization → Set the initial value of the loop control variable

Test-condition → Set the condition to control the loop via loop control variable

Inc/dec → Update the value of the loop control variable

Body of the loop → Statements to be executed or controlled under the for loop

Analyze the for Statement

- Initialization is done first using the assignment statements such as i=1 or count=0. where i and count are known as loop control variable.
- The value of the loop control variable is tested using the test condition. The test condition is a relational expression, such as i < 10 that determines when the loop will exit. If the condition is true the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
- When the body of the loop is executed, the control is transferred back to the **for** statement after evaluating the last statement such as **increment/decrement** and the new value of the control variable is again tested to see whether it satisfies the loop condition or not.

Analyze the for Statement

- When the body of the loop is executed, the control is transferred back to the **for** statement after evaluating the last statement such as **increment/decrement** and the new value of the control variable is again tested to see whether it satisfies the loop condition or not.
- If the condition is satisfied again, then the body of the loop is executed again. This process continues till the value of the control variable fails to satisfy the test condition.
- One of the important points about the for loop is that all the three actions **initialization, test-condition and increment/decrement** are placed within the for statement itself, thus making the user comfortable but it is not mandatory.

Analyze the for Statement

- The for loop in C has several capabilities that are not found in other loop constructs. For example more than one variable can be initialized at a time in the **for** statement separated by a comma. Example: **for (m=1, n=10; m<=n; m++)**
- Like the initialization section the increment section can have more than one part separated by a comma. For example: **for (m=1, n=10; m<=n; m++, n--)**
- The test condition can have compound relation and the testing need not to be limited only to the loop control variable. Example: **for (m=1, n=10; m<10 && n< 20; m++)**
- It is also permissible to use expressions in the assignment statements of initialization and increment/decrement. Example: **for(i=(m+n)/2; i>1; i=i/2)**

Analyze the for Statement

- Another unique and important aspect of for loop is that one or more sections can be omitted, if necessary. However in such cases the semicolons separating the sections must remain. Both initialization and increment/decrement sections can be omitted within the for statement. The initialization can be done before the for statement and increment/decrement can be performed within the body of the loop or even after the body of the loop. Example:

```
i=5;
for(; i<=15;){
        printf("The current value of i=%d", i);
        i=i+1; }
```

Analyze the for Statement

- Again the test condition can also be omitted. If the test condition is not present then the for statement behaves like a infinite loop. Such loop can be broken using break and goto statement. Example:

```
for(;  ;)
        printf("Hello World");
```

- The C compiler does not give an error message if a semicolon is placed at the end of the for statement but it is considered as a null statement. Example:

```
for(i=0;i<=2;i++) ;
```

The While Statement

- The simplest of all the looping structures is the while statement. The basic format of the while statement is :

```
while (test-condition)
        {
        body of the loop;
        }
```

The While Statement

- The while is an entry controlled loop statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body the test condition is once again evaluated and if it is true, the body is executed once again.
- This process of repeated execution of the body continues until the test condition becomes false and the control is transferred out of the loop. On exit the program continues with the statement immediately following the body of the loop.
- The important thing to notice that if the test condition is false then the controlled statements are not executed not even once unlike the **do while** loop.

The Do While Statement

- On some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled by using **do while** loop. It has the structure:

```
do
    {
    body of the loop;
}
while(test condition);
```

Comparison of Loops

Topics	For loop	While loop	Do … While loop
Initialization of condition variable	Before or within the parenthesis of the loop.	Before the loop.	Before or in the body of the loop.
Test Condition	Before the body of the loop.	Before the body of the loop.	After the body of the loop.
Updating the condition variable	After the first execution.	After the first execution.	After the first execution.
Loop Type	Entry controlled loop	Entry controlled loop	Exit controlled loop
Loop Variable	Counter	Counter	Sentinel and Counter

Selecting a Loop

- It is the programmer's choice to use which loop depending on the type of the problem and how he want to solve it. But still there are some strategy that are used while choosing a loop to solve a problem:
- Analyze the problem and see whether it required a pre-test or post-test loop. If it requires a post-test loop then we can use only one loop that is **do while** loop**.** If it requires a pre- test loop then we have two choices **for** loop and **while** loop.
- Decide whether the loop termination requires counter based control or sentinel based control. Use **for** loop if the counter based control is necessary. Use **while** loop if the sentinel based control is required.

Jumping out a Loop

- C permits a jump from one statement to another within a loop as well as a jump out of a loop. The **break** and **goto** statements can be used to jump out of a loop.
- When a **break** statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the **break** would only exit from the loop containing it. That is the **break** will exit only a single loop.
- While a **goto** statement can transfer the control to any place in the program. It is useful to provide branching within a loop. Another important use of **goto** is to exit from deeply nested loops when an error occurs.