SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

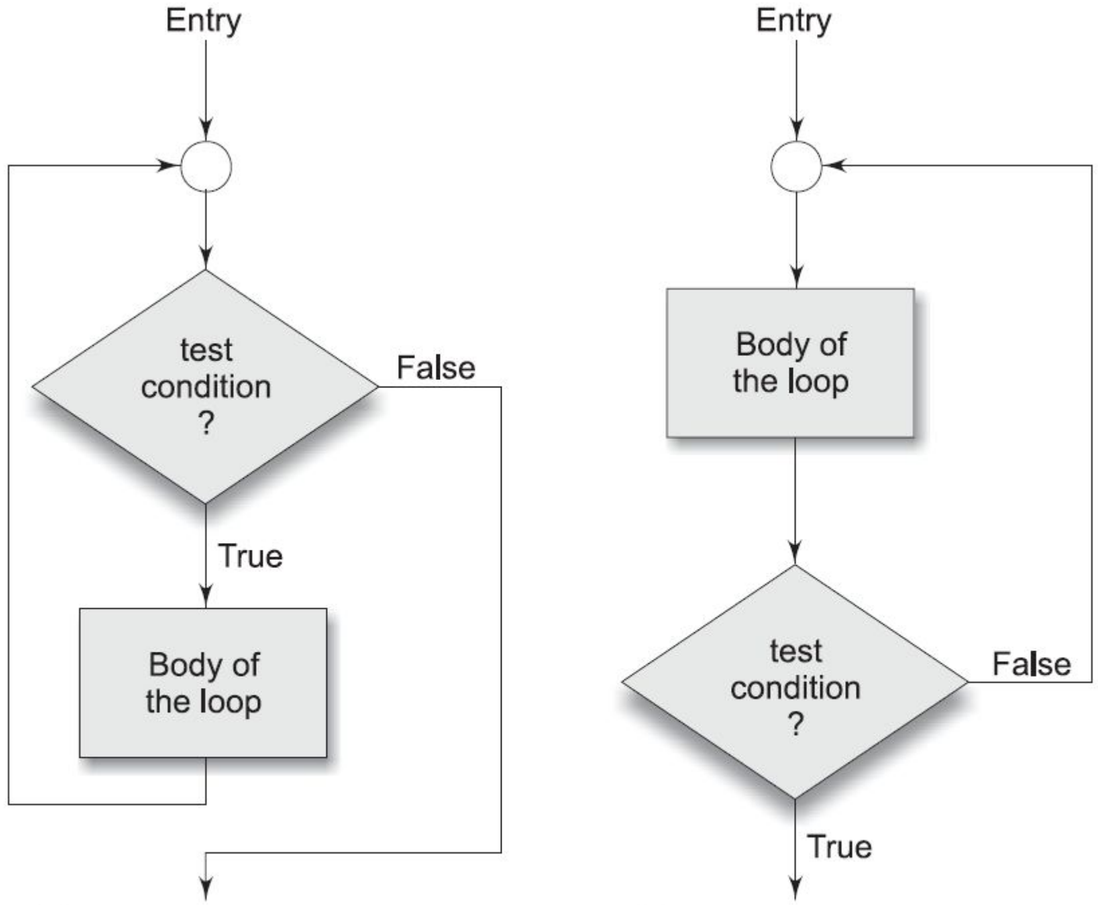
UNIT 2 – C Programming Basics

TOPIC 8 – Decision Making and Looping

INTRODUCTION

- We have seen that it is possible to execute a segment of a program repeatedly by introducing a **counter** and later testing it using the **if** statement.
- While this method is quite satisfactory for all practical purposes, we need to initialize and increment a counter and test its value at an appropriate place in the program for the completion of the loop.
- In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied.
- A program loop therefore consists of **two** segments:
  - 1. Known as the **body of the loop** and
  - 2. 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**

LOOP CONTROL STRUCTURES



Loop control structures

LOOP CONTROL STRUCTURES

- Depending on the position of the **control** statement in the loop, a control structure may be classified either as
  - Entry-controlled loop or
  - Exit-controlled loop.
- Flowcharts illustrates these structures.
- In the **entry-controlled** loop, the control conditions are tested **before** the start of the loop execution.
  - If the conditions are **not satisfied**, then the body of the loop will not be executed.
- In an **exit-controlled loop**, the test is performed at the **end** of the body of the loop and therefore the body is executed unconditionally for the first time.
- The entry-controlled and exit-controlled loops are also known as
  - Pre-test loops &
  - Post-test loops.

LOOP CONTROL STRUCTURES

- The test conditions should be carefully stated in order to perform the desired number of loop executions.
- It is assumed that the test condition will eventually transfer the control out of the loop.
- In case, due to some reason it does not do so, the control sets up an **infinite** loop and the body is executed over and over again.

- A looping process, in general, would include the following four steps:
  1. Setting and **initialization** of a condition variable.
  2. **Execution** of the statements in the loop.
  3. **Test** for a specified value of the condition variable for execution of the loop.
  4. **Incrementing** or updating the condition variable.
- The test may be either to determine whether the loop has been repeated the specified number of times or to determine whether a particular condition has been met.

LOOP CONTROL STRUCTURES

- The C language provides for **three** constructs for performing loop operations.
- They are:
  1. The **while** statement.
  2. The **do** statement.
  3. The **for** statement.

Difference Between Entry Controlled and Exit Controlled Loops

Entry Controlled	Exit Controlled
Condition is checked at the entry of the loop	Condition is checked at the exit of the loop
If condition is initially false, the loop never executes **i=1;** **while(i==0)** **{** **System.out.println("In While loop");** **}** **System.out.println("out of the loop");** Output: **Out of the loop**	If condition is initially false, then also the loop executes at least once **i=1;** **do** **{** **System.out.println("In While loop");** **} while(i==0);** **System.out.println("out of the loop");** Output: **In while loop** **Out of the loop**
Example- for, while	Example – do-while

SENTINEL LOOPS

- Based on the nature of control variable and the kind of value assigned to it for testing the control expression, the loops may be classified into following **two** general categories:
  1. Counter-controlled loops
  2. Sentinel-controlled loops

Counter-controlled loops:

- When we know in advance exactly how many times the loop will be executed, we use a counter controlled loop.
- We use a control variable known as counter.
- The counter must be initialized, tested and updated properly for the desired loop operations.
- A counter-controlled loop is sometimes called **definite repetition** loop.

Sentinel-controlled Loop:

- In a sentinel-controlled loop, a special value called a sentinel value is used to change the loop control expression from true to false.
- For example, when reading data we may indicate the “end of data” by a special value, like –1 and 999.
- The control variable is called sentinel variable.
- A sentinel-controlled loop is often called **indefinite** repetition loop because the number of repetitions is not known before the loop begins executing.

THE WHILE STATEMENT

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

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

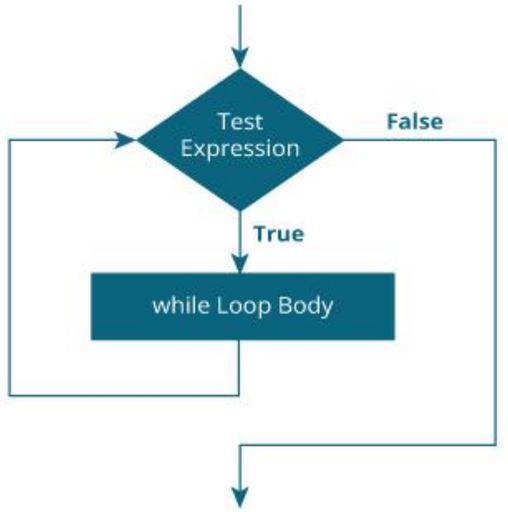
- The while is an **entry-controlled** loop statement.
- The test-condition is evaluated and if the condition is **true**, then the body of the loop is executed.
- After execution of the body, the test-condition is once again evaluated and if it is true, the body is executed once again.
- This process of repeated execution of the body continues until the test-condition finally becomes false and the control is transferred out of the loop.
- On exit, the program continues with the statement immediately after the body of the loop.
- The body of the loop may have one or more statements.

THE WHILE STATEMENT

```
// Print numbers from 1 to 5
#include <stdio.h>
int main()
{
    int i = 1;
    while (i <= 5)
    {
        printf("%d\n", i);
        ++i;
    }
    return 0;
}
```

Output:

1
2
3
4
5




```
#include<stdio.h>
main()
{
   int x,ctr;
   x=0;
   printf("Input number of rows :");
   scanf("%d",&ctr);
   while(x<ctr)
     {
     printf("*");
     x++;
     }
}
```

suppose ctr = 5

	x	output
is x<ctr ?	0	*
is x<ctr ?	1	**
is x<ctr ?	2	***
is x<ctr ?	3	****
is x<ctr ?	4	*****
is x<ctr ?	5	exit from loop

THE DO (While) STATEMENT

- The **while** loop construct that we have discussed in the previous section, makes a test of **condition before the loop is executed**.
- Therefore, the body of the loop **may not be executed** at all if the condition is not satisfied at the very first attempt.
- On some occasions it might be necessary to execute the body of the loop before the test is performed.
- Such situations can be handled with the help of the do statement.
- This takes the form:

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

- On reaching the do statement, the program proceeds to evaluate the body of the loop fi rst. At the end
- of the loop, the test-condition in the while statement is evaluated. If the condition is true, the program

THE DO (While) STATEMENT

- On reaching the **do** statement, the program proceeds to evaluate the body of the loop first.
- At the end of the loop, the test-condition in the while statement is evaluated.
- If the condition is true, the program continues to evaluate the body of the loop once again.
- This process continues as long as the condition is **true**.
- When the condition becomes **false**, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.
- Since the test-condition is evaluated at the bottom of the loop, the **do...while** construct provides an exit controlled loop and therefore the body of the loop is always **executed at least once**.

- Note:

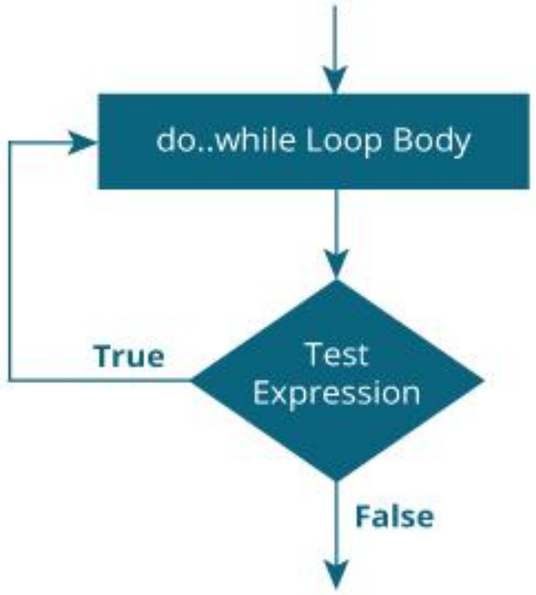
- **do...while** statement is an **exit controlled** loop statement.

THE DO (While) STATEMENT

```
// Program to add numbers until the user enters zero
#include <stdio.h>
int main()
{
    double number, sum = 0;
    // the body of the loop is executed at least once
    do
    {
        printf("Enter a number: ");
        scanf("%lf", &number);
        sum += number;
    }
    while(number != 0.0);
    printf("Sum = %.lf", sum);
    return 0;
}
```

Output:

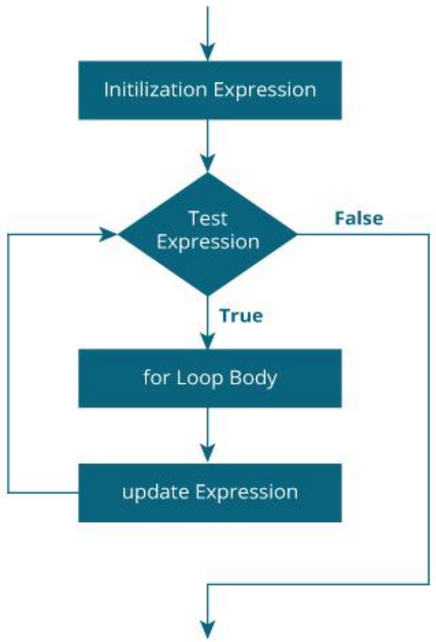
```
Enter a number:1.5
Enter a number:2.4
Enter a number:-3.4
Enter a number:4.2
Enter a number:0
Sum = 4.70
```



THE FOR STATEMENT

- The for loop is another **entry-controlled** loop that provides a more concise loop control structure.
- The general form of the for loop is

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




```c
// Print numbers from 1 to 10
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

Output

```
1 2 3 4 5 6 7 8 9 10
```

THE FOR STATEMENT

- The execution of the **for** statement is as follows:

- 1. **Initialization** of the control variables is done first, using assignment statements such as i = 1 and count = 0.
- The variables i and count are known as **loop-control variables**.

- 2. The value of the 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.

- 3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop.
- Now, the control variable is incremented using an assignment statement such as i = i+1 and the new value of the control variable is again tested to see whether it satisfies the loop condition.
- If the condition is satisfied, the body of the loop is again executed.
- This process continues till the value of the control variable fails to satisfy the test condition.

THE FOR STATEMENT

- In simple Words

- The **initialization** statement is executed only once.
- Then, the **test expression** is evaluated.
- If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements **inside the body of for loop are executed**, and the update expression is updated.
- Again the test expression is evaluated.
- This process goes on until the test expression is false.
- When the test expression is false, the loop terminates.

THE FOR STATEMENT

```c
// 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, count, sum = 0;

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

    // for loop terminates when num is less than count
    for(count = 1; count <= num; ++count)
    {
        sum += count;
    }

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

    return 0;
}
```

Output

```
Enter a positive integer: 10
Sum = 55
```

JUMPS IN LOOPS

- Loops perform a set of operations repeatedly until the control variable fails to satisfy the test-condition.
- The number of times a loop is repeated is decided in advance and the test condition is written to achieve this.
- Sometimes, when executing a loop it becomes **desirable to skip a part of the loop** or to leave the loop as soon as a certain condition occurs.

- For example, consider the case of searching for a particular name in a list containing, say, 100 names.
- A program loop written for reading and testing the names 100 times must be terminated as soon as the desired name is found.

- C permits a **jump** from one statement to another within a loop as well as a jump out of a loop.

JUMPS IN LOOPS

- Break Statement
- 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.**