• Chapter 12 : Loop Constructs

12.1 Overview

Q : 12-01-01 : Describe Iteration and Loop ?

Answer :

Iteration or Loop : There are problems whose solution may require executing a ,statement or a set of statements repeatedly. We need a structure that would allow repeating a set of statements up to fixed number of times or until a certain criterion is satisfied. [Iteration is the third type of program control structure (sequence, selection, iteration), and the repetition of statements in a program is called a loop]. Loop control statements are while, do-while, and for.

12.2 While Statement

Q : 12-02-01 : Describe While Statement with Flowchart and example ?

Answer :

While Statement : [The while loop keeps repeating associated statements until the specified
condition becomes false]. This is useful where the programmer does not know in advance how many times the loop will be traversed. The syntax of the while statement is :

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

The condition in the while loop controls the loop iteration. The statements, which are executed when the given condition is true, form the body of the loop. If the condition is true, the body of the loop is executed. As soon as it becomes false, the loop terminates before starting next iteration.

Flowchart of The While Loop :

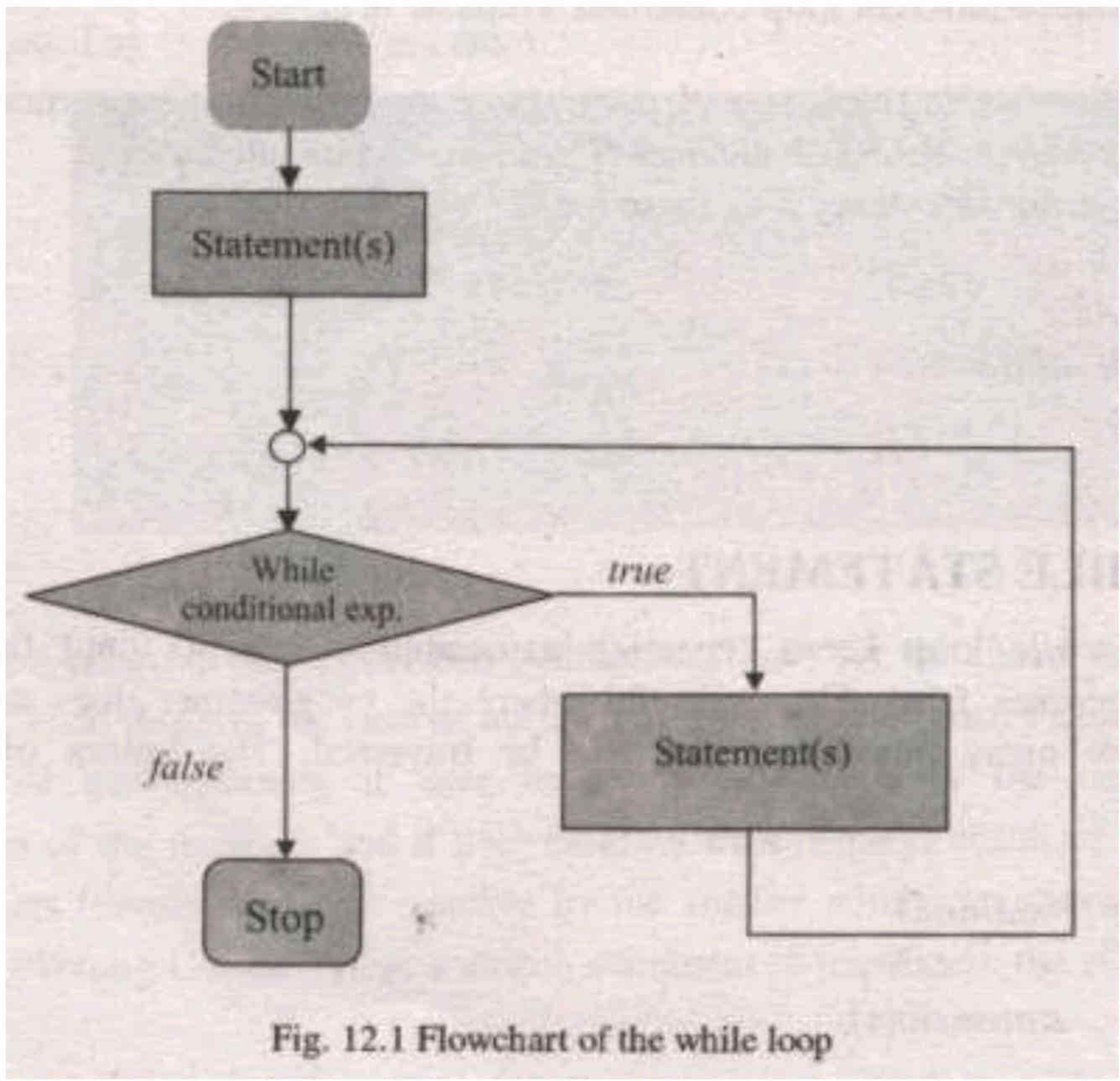


Fig. 12.1 Flowchart of the while loop

Example : Write a program to print digits from 1 to 10 using while loop.

```
#include<stdio.h>
void main(void)
{
        int count ;
        count = 1 ;
        while (count <= 10)
        {
                printf("%d\n", count) ;
                count = count + 1 ;
        }
}
```

This is a simple program which demands iterative solution. It does not make sense to use ten printf statements to print ten digits; if so, what if we have to print digits from 1 to 1000 ? Should we write one thousand printf statements to accomplish the task ? Certainly not; the right way to come up to the solution is to use a loop, which would execute ten times. Each time the loop executes, a number (next) is printed, which is incremented by one for every iteration until the required list of numbers is printed. In this program, we use a variable count which is initialized by 1. The condition (count <= 10) depends on the value of this variable. Until the condition is TRUE, the control will enter the body of the loop, and as soon as it becomes FALSE, the control will exit from the loop and will jump to the next statement to the body of the loop. First time, when the condition is checked, it is found true as the value of count which is one, is less than ten. The control enters the body of the loop and the number "1" is printed. The next line of code increments the value of count by one, which becomes "2". After that, the control will immediately jump to the while statement where again the condition is tested which is still found TRUE, as 2 is less than 10. The control again enters the body of the loop, and the number "2" is printed. The value of the variable count again increases by one and becomes "3". The control again transfers

to the while statement. This process continues until the value of count becomes "11", making the condition FALSE. When the condition becomes FALSE, the control will exit from the loop.

Important Note : The count is the loop control variable. A variable whose value controls the
number of iterations is known as loop control variable. The compound statement, which is enclosed in braces, is the body of the loop. In while loop; the loop control variable is always initialized outside the loop and is incremented or decremented inside the loop body.

12.3 Do-While Loop

Q : 12-03-01 : Describe Do-While Statement with Flowchart and example ?

Answer :

While Statement / Loop : This is very similar to the while 1oop except that the test occurs at the end of the loop body. This guarantees that the loop is executed at least once. This loop is frequently used where data is to be read; the test then verifies the data, and loops back to read again if it was unacceptable. The syntax of the do-while statement is :

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

The important point about this loop is that unlike while loop, it ends with a semicolon. Omitting the semicolon will cause a syntax error. Let us re-write the program in while loop example.

Example 1 : Write a program to print digits from 1 to 10 using do-while loop.

```
#include<stdio.h>
void main(void)
{
        int count ;
        count = 1 ;
        do
        {
                printf("%d\n", count) ;
                count = count + 1 ;
        } while (count <= 10) ;
}
```

Here, we achieve the same objective but in a different way. The keyword do let the program flow to move into the body of the loop without checking any test condition. It means, whatever is written in the loop body always will be executed at least once. At the completion of execution of the body of the loop, the test condition is checked. If it is found true, the control is transferred to the first statement in the body of the loop, and if the condition is evaluated to false, the loop terminates immediately and the control moves to the very next instruction outside the loop. The do-while loop is of great importance in situations where we need to execute certain statements at least

once.

Example 2 : Your telephone connection may be in any of two states working (W) or dead (D). Write a program that reads the current state of the telephone line; the user should enter W for working state and D for dead state. Any input other than W or D, will be considered invalid. Force the user to enter a valid input value. This could be achieved by using a do-while loop.

```
#include<stdio.h>
void main(void)
{
        char    state ;
        do
        {
                printf("\nPlease Enter Current State Working (W) or Dead
                (D) => ") ;
                scanf("%c", &state) ;
        } while (state != 'w' && state != 'W' && state != 'd' && state !=
        'D') ;
        if (state == 'w' || state == 'W')
                printf("\nThe State of your phone is WORKING") ;
        else if (state == 'd' || state == 'D')
                printf("\nThe State of your phone is DEAD") ;
}
```

This program demonstrates a scenario where an invalid input is not processed, until the user enters a valid input (d or w or D or W), the program repeatedly shows him (or her) the message for the valid input to be entered. Here, the key point is the correct understanding of the test condition (state != 'w' && state != 'W' && state != 'd' && state != 'D'). It is a compound condition which is comprised of four sub conditions.

Important Note : It should be noted that if two or more conditions are combined using logical AND operator to form a compound condition, the compound condition will be true only if all the sub conditions are true and if any of the sub conditions is false, the compound condition evaluates to false. Therefore, the user enters an invalid input (suppose e), all four sub conditions evaluate to true (because e is not equal to w, W, d or D). Therefore the compound condition also evaluates to true and the control flow returns back to the start of loop body. This process continues until the user enters a w or W or d or D, one of the sub conditions evaluates to false causing the compound condition to be evaluated to false and the control flow exits the loop body.

while vs do-while

- In *while* loop, the *body of the loop* may or may not execute depending on the evaluation of the test condition.
- In *do-while* loop, first the *body of the loop* is executed and then the test condition is checked. Hence it always executed at least once.

12.3a FOR Loop – A Poetic Structure

Q : 12-03a-01 : Describe FOR Statement, its Execution Sequence, with flowchart and example ?

Answer :

FOR Statement / Loop : The for statement is another way of implementing loops in C. Because of its flexibility, most programmers prefer the "for" statement to implement loops. The syntax of the for loop is :

```
for (initialization expression ;  test condition ;  increment / decrement expression)
{
    statement(s) ; // loop body
}
```

There are three expressions in for loop statement, these are :

Initialization of the loop control variable.
Test condition.
Change (increment or decrement) of the loop control variable.

```
for (stmt1;  stmt2 ;  stmt3)
    stmt4 ;
```

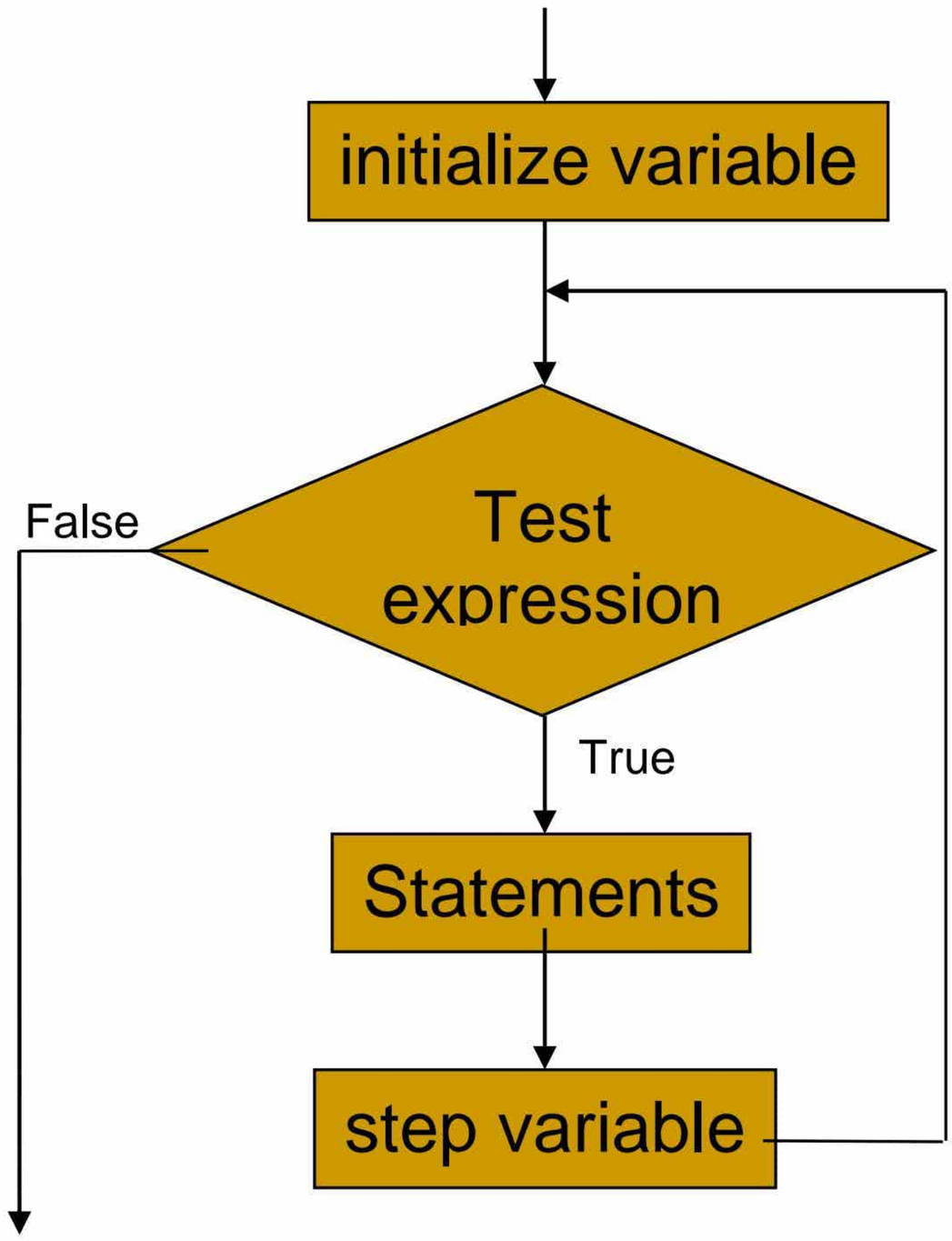
Execution Sequence : The four statements / expressions are :

Stmt1 : Initialization Statement(s) (may or may not exist).
Stmt2 : Test / Boolean Expression (may or may not exist).
Stmt3 : Stepping Statement(s) (may or may not exist).
Stmt4 : Body Block / Body Statement (may or may not exist).

On Entry To The Loop, Sequence of EXECUTION is :

Step 1 : Stmt1(if exists) shall be executed ONLY ONCE.
Step 2 : Stmt2(if exists) shall be evaluated for a BOOLEAN value, if ZERO go to Step 5.
Step 3 : Stmt4(if exists) shall be executed.
Step 4 : Stmt3(if exists) shall be executed and go to Step 2.
Step 5 : Exit THE "FOR" LOOP.

Important Note : If stmt3 is not changed / stepped, the loop continues for ever; such a loop is called infinite loop.



The initialization expression is executed in only the first iteration. Then the loop condition is tested. If it is true, the statements in the body of the loop are executed. After execution of the body of the loop, the increment / decrement expression is evaluated. It is very important to note that the initialization expression is only executed for the first iteration. For second and next iterations, the loop condition is tested, if it is true then the body of the loop is executed and then increment / decrement expression is evaluated. After evaluation "increment / decrement" expression, the test condition is checked again and if it is true then the body of the loop is executed. This process continues as long as the loop condition is true. When this condition is found to be false, the "for" loop is terminated and the control transfers to the next statement following the "for" loop. Usually, we increment or decrement the loop control variable in the increment / decrement expression.

Example : Write a program to print digits from 1 to 10 using "for" loop.

```
#include<stdio.h>
void main(void)
{
int     count ;
for (count = 1 ;      count <= 10 ;        count++)
        printf ("%d\n", count) ;
}
```

There are three expressions in for loop statement, separated by semicolons. Any of these can be omitted but we must be aware of the consequences as : for (; ;) ; is a legal statement and will compile and run but will never terminate; this is called INFINITE LOOP.

12.4 Nested Loop

Q : 12-04-01 : Describe Nested Loop Statement with example ?

Answer :

Nested Loop : Nested loop means a loop inside the body of another loop. Nesting can be done up to any level. But, as the level of nesting increases, the complexity of the nested loop also increases. There is no restriction on the type of loops (while, do while, or for) that may be placed in the body of other loops.

Example : Write a program that will print asterisks (*) according to the given pattern :

```
*******
******
*****
****
***
**
*
```

```
#include<stdio.h>
void main(void)
{
        int     outer, inner ;
        for (outer = 7 ;       outer >= 1 ; outer--)
        {
                inner = 1 ;
                while ( inner <= outer )
                {
                        printf ( "*" ) ;
                        inner++ ;
                }
                printf ( "\n" ) ;
        }
}
```

In this program, a while loop is used inside the body of for loop, which shows a nested loop. The outer loop is controlled by the loop control variable i.e., outer. The outer loop is executed seven times. For each iteration of the outer loop, the inner loop executes until the value of the inner loop control variable i.e., inner is less than or equal to the value of the variable outer. It should be noted that each time a new iteration for the outer loop starts, the variables used in the inner loop are re-initialized and re-processed.

For the first iteration of the outer loop, the variable 'outer' is initialized to 7, and in all next iterations it is decremented by 1. This process continues until the value of the variable is greater than or equal to 1. For the first iteration of the outer loop, the inner loop executes seven times, and for the 2nd iteration it executes six times, similarly for the last seventh iteration, the inner loop executes just one time. Each time when the inner loop is terminated, the statement printf ("\n") moves the cursor to the start of the new line.

Q : 12-04-02 : Define Sentinel Value ? Write an example program to show data entry by a user to calculate average marks of a student ?

Answer :

Sentinel Value : [Sentinel Value is an end marker that follows the last item in a list of items]. Many programs require a list of items to he entered by the user. Often, don't know how many items the list will have. For example, to find the average marks of a class, we have to input the marks of every student of the class. Similarly, calculate the sum of a series, we have to input the list of numbers in the series. There are so many other situations where the solution demands to enter a list of items process. Loops are very useful to develop solutions for such problems. Each time the loop body is repeated, one or more data items are input. But, often we don't know how many data items will he input by the user. Therefore, we must find some way to signal the program to stop reading and processing new data. One way to do this is to instruct the user to enter a unique data value called a sentinel value, after the last data item. The loop condition tests each data item art causes loop exit when the sentinel value is read. Choose the sentinel value carefully; it must be a value that could not normally occur as data. The general form of a sentinel controlled loop is :

Get the first line of data.
While the sentinel value has not been encountered.
Process the data line.
Get another line of data.

Example : Write a program to find the average marks of the students in a class.

```
#include<stdio.h>
void main (void)
{
        int sum = 0, marks = 0, total_students = 0;
        float average ;
        do
        {
                printf("Enter marks of the student (or any -ve number to quit) => ") ;
                scanf("%d", &marks) ;
                if (marks >= 0)
                {
                        total_students++ ;
                        sum += marks ;
                }
        } while (marks > 0) ;
        if (total_students > 0)
        {
                average = sum / (float) total_students ;
                printf("The Average Marks of The Class are : %f\n", average) ;
        }
        else
                printf("Enter Marks of at least one student to Calculate Average\n") ;
}
```

This program demonstrates a typical implementation of sentinel loop. Size of the class does not matter, whatever it is, the average will be calculated in the same way. Here any negative number may act as the sentinel value because no student can have negative marks. However, zero would not be a wise option because there can be a student with zero marks. The program reads the marks until the user enters a negative number. For every valid input (zero and +ve numbers) the control switches to the body of the while loop. In the loop body, the total_students is incremented by one and the sum is accumulated. As soon as a negative number is entered, the sentinel while loop is terminated. The next line to the end of the while loop is an if statement, which checks the count for total students i.e., total_students to ensure that the marks of at least one student have been entered. Omitting this if statement may crash the program. It is because of the formula for calculating average where the sum is divided by total_students. When marks of any students are not entered, the value of total_students is zero and calculating average for zero students will result in a runtime error of division by zero. So, to avoid this possible error first the value of the variable total_students is cheked; if its greater than zero then the average is calculated otherwise a message is shown to the user to enter the marks of at least one student.
Now, notice the average formula : average = sum / (float)total_students ;
We have used the keyword float in parenthesis before the variable total_students. The reason is that both the variables sum and total_students are integers. So, their division will be integral division in which the fractional part is truncated. Hence, the result will not be accurate. Writing float in parenthesis (float) before the integer variable name (i.e. total_students) causes the variable to temporarily act as a float variable for this particular calculation. It is done to preserve the fractional part in the result. The integer variable (total_students) will act as an integer for all other calculations. The effect of this change is strictly associated with that particular calculation. This phenomenon is known as type casting.

12.5 GOTO Statement

Q : 12-05-01 : Describe goto Statement ? Explain it with Example Program ?

Answer :

The goto statement performs an unconditional transfer of control to the named label. The label must be in the same function. A label is meaningful only to a statement; in any other context, the labeled statement is executed without regard to the label. The general form of the goto statement is as :

```
goto label ;
label: statement ;
```

Example : Write a program to calculate the square root of a positive number (handle negative numbers properly).

```
#include<math.h>
#include<stdio.h>
void main (void)
{
```

```
                float num ;
                positive:
                printf("Please Enter a positive number => ") ;
                scanf("%f", &num) ;
                if (num < 0)
                        goto positive ;
                else
                        printf("Square Root of %0.2f is %0.2f", num,
                sqrt(num)) ;
        }
```

If the user enters a negative number, the control transfers to the label positive.

Important Note : Use of goto statement is not appreciated in C.

Exercise 12

Q-9. Write a program that inputs a number and displays the message "Prime Number" if it is a prime number, otherwise displays "Not a Prime Number".

Answer :

```
#include<math.h>
#include<stdio.h>
void main (void)
{
        unsigned int num ;
        printf("Please Enter a Positive Number To Check
        Primality => ") ;
        scanf("%f", &num) ;
        if (num % 2 == 0)
        {
                printf("The Number %d is NOT Prime", num);
                exit(0);
        }
        else
        {
                for (int i = 3 ; i < sqrt(num) ; i += 2)
                {
                        if (num % i == 0)
                        {
                                printf("The Number %d is NOT
                Prime", num);
                                exit(0);
                        }
                }
                printf("The Number %d is Prime", num);
        }
}
```

Q-10. Write a program that displays the first 15 even numbers.

Answer :

```
#include<stdio.h>
void main (void)
{
        for (int i = 1 ; i < 16 ; i ++)
        {
                printf("\n%d", i*2);
        }
```

Q-11. Write a program that inputs a number, and displays its table according to the following format:

Suppose the number entered is 5, the output will be as follows :

```
5 *  1  =  5
5 *  2  = 10
5 *  3  = 15
.
.
5 * 10 = 50
```

Answer :

```
#include<stdio.h>
void main (void)
{
        unsigned int num ;
        printf("Please Enter a Positive Number To Print Table
        => ") ;
        scanf("%f", &num) ;
        for (int i = 1 ; i < 11 ; i ++)
        {
                printf("\n%d x %d = %d", i, num, i * num) ;
        }
}
```

Q-12. Write a program using do-while loop that repeatedly prompts for and takes input until a value in the range 0 through 15 inclusive is input. The program should add all the values before exiting the loop and displays their sum at the end.

Answer :

```
#include<stdio.h>
void main (void)
{
        int num = 0 , total = 0 ;
        do
```

```
                {
                        printf("\nPlease Enter a Number (0 To 15) =>
                ") ;
                        scanf("%f", &num) ;
                        total += num ;
                } while (num < 0 || num > 15) ;
                printf("\nThe Total = %d", total) ;
        }
```

Q-13. Write a program that produces the following output :

```
0
0       1
0       1       2
0       1       2       3
0       1       2       3       4
0       1       2       3       4       5
```

Answer :

```
#include<stdio.h>
void main (void)
{
        for (int i = 0 ; i < 10 ; i++)
        {
                printf("\n") ;
                for (int j = 0 ; j <= i ; j++)
                        printf("\t%d", j) ;
        }
}
```

Q-14. Write a program the produces the following output :

```
0       1
1       2
2       4
3       8
4       16
5       32
6       64
```

Answer :

```
#include<math.h>
#include<stdio.h>
void main (void)
{
        for (int i = 0 ; i <= 6 ; i++)
                printf("\n\t %d \t %d", i, pow(2, i)) ;
}
```