Example 6.1

A program to evaluate the equation

$$y = x^n$$

when n is a non-negative integer, is given in Fig.6.2

The variable **y** is initialized to 1 and then multiplied by **x**, n times using the **while** loop. The loop control variable, **count** is initialized outside the loop and incremented inside the loop. When the value of **count** becomes greater than **n**, the control exists the loop.

EXAMPLE OF **while** STATEMENT

Program

```
main()
{
    int count, n;
    float x, y;

    printf("Enter the values of x and n : ");
    scanf("%f %d", &x, &n);
    y = 1.0;
    count = 1;                /* Initialisation */

    /* LOOP BEGINs */

    while ( count <= n)       /* Testing */
    {
         y = y*x;
         count++;            /* Incrementing */
    }
    /* END OF LOOP */
    printf("\nx = %f; n = %d; x to power n = %f\n",x,n,y);
}
```

Output

```
Enter the values of x and n : 2.5  4
x = 2.500000; n = 4; x to power n = 39.062500

Enter the values of x and n : 0.5  4
x = 0.500000; n = 4; x to power n = 0.062500
```

***Fig.6.2** Program to compute **x** to the power n using **while** loop*

Example 6.2

A program to print the multiplication table from 1 x 1 to 12 x 10 as shown below is given in Fig. 6.3.

1	2	3	4	 10
2	4	6	8	 20
3	6	9	12	 30
4				 40
-				-
-				-
-				-
12	.	.	.	 120

This program contains two **do.... while** loops in nested form. The outer loop is controlled by the variable **row** and executed 12 times. The inner loop is controlled by the variable **column** and is executed 10 times, each time the outer loop is executed. That is, the inner loop is executed a total of 120 times, each time printing a value in the table.

PRINTING OF *MULTIPLICATION* ***TABLE***

Program:

```
#define COLMAX 10
#define ROWMAX 12

main()
{
    int row,column, y;

    row = 1;
    printf("          MULTIPLICATION TABLE           \n");
    printf("-----------------------------------------\n");
    do    /*......OUTER LOOP BEGINS........*/
    {
         column = 1;

         do   /*.......INNER LOOP BEGINS.......*/
         {
              y = row * column;
              printf("%4d", y);
              column = column + 1;
         }
         while (column <= COLMAX); /*... INNER LOOP ENDS ...*/

         printf("\n");
         row = row + 1;
    }
    while (row <= ROWMAX);/*.....   OUTER LOOP ENDS   .....*/
```

```
    printf("-----------------------------------------\n");
}
```

Output

```
                    MULTIPLICATION TABLE
-----------------------------------------------------
  1     2     3     4     5     6     7     8     9     10
  2     4     6     8     10    12    14    16    18    20
  3     6     9     12    15    18    21    24    27    30
  4     8     12    16    20    24    28    32    36    40
  5     10    15    20    25    30    35    40    45    50
  6     12    18    24    30    36    42    48    54    60
  7     14    21    28    35    42    49    56    63    70
  8     16    24    32    40    48    56    64    72    80
  9     18    27    36    45    54    63    72    81    90
  10    20    30    40    50    60    70    80    90    100
  11    22    33    44    55    66    77    88    99    110
  12    24    36    48    60    72    84    96    108   120
-----------------------------------------------------
```

Fig.6.3 *Printing of a multiplication table using* ***do...while loop***

Example 6.3

The program in Fig.6.4 uses a **for** loop to print the "Powers of 2" table for the power 0 to 20, both positive and negative.

The program evaluates the value

$$p = 2^n$$

successively by multiplying 2 by itself n times.

$$q = 2^{-n} = \frac{1}{p}$$

Note that we have declared **p** as a ***long int*** and **q** as a **double**.

Additional Features of for Loop

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. The statements

```
p = 1;
for (n=0; n<17; ++n)
```

can be rewritten as

```
for (p=1, n=0; n<17; ++n)
```

USE OF **for** LOOP

Program:

```
main()
{
    long int p;
    int      n;
    double   q;
    printf("-------------------------------------------\n");
    printf(" 2 to power n       n       2 to power -n\n");
    printf("-------------------------------------------\n");
    p = 1;
    for (n = 0; n < 21 ; ++n)    /* LOOP BEGINS */
    {
        if (n == 0)
           p = 1;
        else
           p = p * 2;
        q = 1.0/(double)p ;
        printf("%10ld %10d %20.12lf\n", p, n, q);
    }                            /* LOOP ENDS   */
    printf("-------------------------------------------\n");
}
```

Output

```
---------------------------------------------
     2 to power n   n       2 to power -n
---------------------------------------------
         1          0       1.000000000000
         2          1       0.500000000000
         4          2       0.250000000000
         8          3       0.125000000000
        16          4       0.062500000000
        32          5       0.031250000000
        64          6       0.015625000000
       128          7       0.007812500000
       256          8       0.003906250000
       512          9       0.001953125000
      1024         10       0.000976562500
      2048         11       0.000488281250
      4096         12       0.000244140625
      8192         13       0.000122070313
     16384         14       0.000061035156
     32768         15       0.000030517578
     65536         16       0.000015258789
    131072         17       0.000007629395
    262144         18       0.000003814697
    524288         19       0.000001907349
   1048576         20       0.000000953674
---------------------------------------------
```

Fig.6.4 *Program to print* ***'Power of 2'*** *table using for loop*

Example 6.4

A class of **n** students take an annual examination in **m** subjects. A program to read the marks obtained by each student in various subjects and to compute and print the total marks obtained by each of them is given in Fig.6.5.

The program uses two **for** loops, one for controlling the number of students and the other for controlling the number of subjects. Since both the number of students and the number of subjects are requested by the program, the program may be used for a class of any size and any number of subjects.

The outer loop includes three parts:

(1) reading of roll-numbers of students, one after another,

(2) inner loop, where the marks are read and totaled for each student, and

(3) printing of total marks and declaration of grades.

ILLUSTRATION OF **NESTED** LOOPS

Program

```
#define FIRST   360
#define SECOND  240
main()
{
    int n, m, i, j,
        roll_number, marks, total;
    printf("Enter number of students and subjects\n");
    scanf("%d %d", &n, &m);
    printf("\n");
    for (i = 1; i <= n ; ++i)
    {
        printf("Enter roll_number : ");
        scanf("%d", &roll_number);
        total = 0 ;
        printf("\nEnter marks of %d subjects for ROLL NO %d\n",
                  m,roll_number);
        for (j = 1; j <= m; j++)
        {
            scanf("%d", &marks);
            total = total + marks;
        }
        printf("TOTAL MARKS = %d ", total);
        if (total >= FIRST)
            printf("( First Division )\n\n");
        else if (total >= SECOND)
                printf("( Second Division )\n\n");
             else
                printf("( ***  F A I L  *** )\n\n");
    }
}
```

Output

```
Enter number of students and subjects
 3    6
Enter roll_number : 8701
Enter marks of 6 subjects for ROLL NO 8701
81  75  83  45  61  59
TOTAL MARKS = 404 ( First Division )

Enter roll_number : 8702
Enter marks of 6 subjects for ROLL NO 8702
51  49  55  47  65  41
TOTAL MARKS = 308 ( Second Division )

Enter roll_number : 8704
Enter marks of 6 subjects for ROLL NO 8704
40  19  31  47  39  25
TOTAL MARKS = 201 ( ***  F A I L  *** )
```

Fig.6.5** Illustration of **nested for loops

Example 6.5

The program in Fig.6.8 illustrates the use of the break statement in a C program.

The program reads a list of positive values and calculates their average. The **for** loop is written to read 1000 values. However, if we want the program to calculate the average of any set of values less than 1000, then we must enter a 'negative' number after the last value in the list, to mark the end of input.

USE OF **break** IN A PROGRAM

Program

```
main()
{
    int m;
    float x, sum, average;

    printf("This program computes the average of a
                set of numbers\n");
    printf("Enter values one after another\n");
    printf("Enter a NEGATIVE number at the end.\n\n");
    sum = 0;
    for (m = 1 ; m <= 1000 ; ++m)
    {
        scanf("%f", &x);
        if (x < 0)
           break;
        sum += x ;
    }
```

```
        average = sum/(float)(m-1);
        printf("\n");
        printf("Number of values = %d\n", m-1);
        printf("Sum              = %f\n", sum);
        printf("Average          = %f\n", average);
    }
```

Output

```
This program computes the average of a set of numbers
Enter values one after another
Enter a NEGATIVE number at the end.

21  23  24  22  26  22  -1

Number of values = 6
Sum              = 138.000000
Average          = 23.000000
```

Fig.6.8 *Use of* ***break*** *in a program*

Example 6.6

A program to evaluate the series

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + ... + x^n$$

for -1 < x < 1 with 0.01 per cent accuracy is given in Fig.6.9. The **goto** statement is used to exit the loop on achieving the desired accuracy.

We have used the **for** statement to perform the repeated addition of each of the terms in the series. Since it is an infinite series, the evaluation of the function is terminated when the term x^n reaches the desired accuracy. The value of n that decides the number of loop operations is not known and therefore we have decided arbitrarily a value of 100, which may or may not result in the desired level of accuracy.

EXAMPLE OF **exit** WITH **goto** STATEMENT

Program

```
#define   LOOP       100
#define   ACCURACY   0.0001
main()
{
```

```
    int n;
    float x, term, sum;

    printf("Input value of x : ");
    scanf("%f", &x);
    sum = 0 ;
    for (term = 1, n = 1 ; n <= LOOP ; ++n)
    {
        sum += term ;
        if (term <= ACCURACY)
           goto output; /* EXIT FROM THE LOOP */
        term *= x ;
    }
    printf("\nFINAL VALUE OF N IS NOT SUFFICIENT\n");
    printf("TO ACHIEVE DESIRED ACCURACY\n");
    goto end;
    output:
    printf("\nEXIT FROM LOOP\n");
    printf("Sum = %f;  No.of terms = %d\n", sum, n);
    end:
    ;       /* Null Statement */
}
```

Output

```
Input value of x : .21
EXIT FROM LOOP
Sum = 1.265800;  No.of terms = 7

Input value of x : .75
EXIT FROM LOOP
Sum = 3.999774;  No.of terms = 34

Input value of x : .99
FINAL VALUE OF N IS NOT SUFFICIENT
TO ACHIEVE DESIRED ACCURACY
```

***Fig.6.9** Use of **goto** to exit from a loop*

Example 6.7

The program in Fig.6.11 illustrates the use of **continue** statement.

The program evaluates the square root of a series of numbers and prints the results. The process stops when the number 9999 is typed in.

In case, the series contains any negative numbers, the process of evaluation of square root should be bypassed for such numbers because the square root of a negative number is not defined. The **continue** statement is used to achieve this. The program also prints a message saying that the number is negative and keeps an account of negative numbers.

The final output includes the number of positive values evaluated and the number of negative items encountered.

USE OF **continue** STATEMENT

Program:

```
#include <math.h>

main()
{
    int count, negative;
    double number, sqroot;

    printf("Enter 9999 to STOP\n");
    count = 0 ;
    negative = 0 ;

    while (count <= 100)
    {
        printf("Enter a number : ");
        scanf("%lf", &number);
        if (number == 9999)
           break;     /* EXIT FROM THE LOOP */
        if (number < 0)
        {
           printf("Number is negative\n\n");
           negative++ ;
           continue;  /* SKIP REST OF THE LOOP */
        }
        sqroot = sqrt(number);
        printf("Number      = %lf\n Square root = %lf\n\n",
                              number, sqroot);
        count++ ;
    }
    printf("Number of items done = %d\n", count);
    printf("\n\nNegative items       = %d\n", negative);
    printf("END OF DATA\n");
}
```

Output

```
Enter 9999 to STOP
Enter a number : 25.0
```

```
Number      = 25.000000
Square root = 5.000000

Enter a number : 40.5
Number      = 40.500000
Square root = 6.363961

Enter a number : -9
Number is negative

Enter a number : 16
Number      = 16.000000
Square root = 4.000000

Enter a number : -14.75
Number is negative

Enter a number : 80
Number      = 80.000000
Square root = 8.944272

Enter a number : 9999
Number of items done = 4
Negative items       = 2
END OF DATA
```

Fig.6.11 *Use of* ***continue*** *statement*