The computer is well suited for applications which have an operation that needs to be repeated many times. Suppose that we want to process the grades of 10 students using the grade-algorithm we used earlier. You would probably say that we should write the code segment 10 times. What if we were computing for 100 students? It would be insane to write the code segments 100 times. What if we are not sure if we will compute the grade for 15 or 50 students? You might say that we should just run the code as many times as we need. However, both approaches are tedious. Since the code for each student is the same, we need a statement in C that will allow the computer to perform the same grade computation several times.

For this reason, we introduce iterative statements. Such statements can either be event-controlled or count-controlled. Later, we will illustrate the difference between the two.

5.1 Event-Controlled Loops

Supposing we want to add several numbers and these numbers are input by the user. It is impractical for us to ask the user how many numbers he wants to add. The desired solution would be to provide a looping structure that allows the user to inform the program when to terminate. This requires loops that are non-deterministic or event-controlled. An event-controlled looping structure continues to loop until some condition is met.

So why don't we tell the user that we will keep on adding numbers he inputs while the number is not 0. The turning point or the change in action is determined by the event of the user inputting 0. Now that we know what event to watch out for, we have to construct our program.

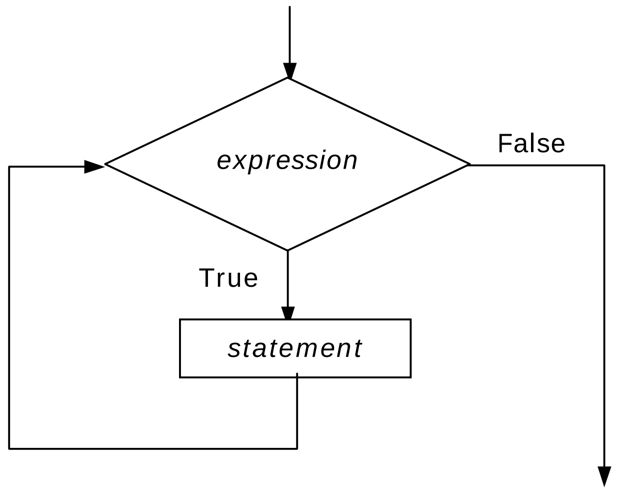
5.1.1 The while Statement

In C, one event-controlled loop is the *while statement*. The syntax is as follows:

Format:

```
     while ( expression )
          statement;
```

The *while statement* continues to execute as long as the expression evaluates to true (or nonzero). This means in plain English "While some condition is true, do some statement". Let's look at the flow of control using a flowchart.



Now, let's apply the while statement in the previous example:

```
main ()
{
     int nInputNum;
     int nTotal = 0;

     scanf( "%d", &nInputNum );

     while ( nInputNum != 0 )
     {
          nTotal += nInputNum; /*accumulate values */
          scanf("%d", &nInputNum); /*get next input */
     }

     printf( "The sum of all input is %d\n", nTotal );
}
```

Note that the only way we could exit the loop is when the user inputs 0. Since the value 0 would cause our loop to terminate, it is referred to as the ***sentinel value***.

5.1.2 The do-while Statement

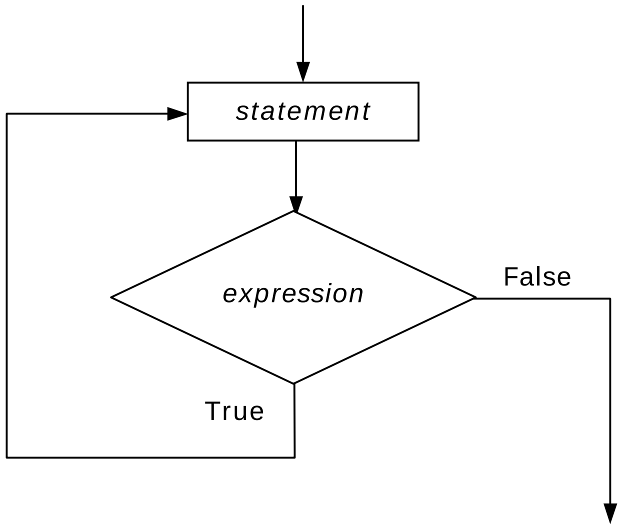
There is another type of event-controlled loop. This one allows us to execute a series of statements before checking the condition. This is the *do-while statement*. The syntax is:

Format:

```
do
{
   statement;
} while ( expression );
```

In the *do-while statement*, the body of the loop is always executed at least once because the condition is not checked until the end of the loop. The loop keeps on executing while the expression evaluates to true (non-zero). When translated in English, it means "do the following statements while the condition is satisfied".

Now, let us use a flowchart to visualize the flow of control in a do-while statement.



Let's apply the *do-while statement* to add a series of input numbers.

```
main ()
{    int nInputNum;
     int nTotal = 0;

     do
     {
          scanf( "%d", &nInputNum );  /*get an input */
          if (nInputNum != 0) /* check input if valid */
             nTotal += nInputNum; /*accumulate values */
     } while ( nInputNum != 0 );

     printf( "The sum of all input is %d\n", nTotal );
}
```

Comparing our while and do-while solution, notice that we did not need to rewrite the scanf statement in the do-while. However, we have to include an if statement to check if the input number is valid before we include that in the computation. But since our sentinel value is 0 and including 0 in the computation qould not change the resulting total, we may rewrite our code to the following:

```
main ()
{
     int nInputNum;
     int nTotal = 0;

     do
     {    scanf( “%d”, &nInputNum );  /*get an input */
          nTotal += nInputNum;   /*accumulate values */
     } while ( nInputNum != 0 );

     printf( “The sum of all input is %d\n”, nTotal );
}
```

Example 1. Write a program that accepts values in feet and converts it into meters. The program will terminate execution if a value less than zero is entered.

```
#define CONVFACT 0.3048;

main ()
{
     double dFeet;

     printf (“Enter value to convert in Feet : “ );
     scanf ( “%lf”, &dFeet );
     while ( dFeet >=0 )
     {
          printf ( “%f ft. is equal to %f  m. \n”,
                  dFeet, (dFeet * CONVFACT) );
          printf (“Enter value to convert in Feet : “);
          scanf ( “%lf”, &dFeet );
     }
     printf (“Thank you ! \n”);
}
```

Sample Dialogue:

```
Enter value to convert in Feet :  10
10 ft. is equal to 3.048 m.
Enter value to convert in Feet :  -3
Thank you !
```

Example 2. Write a program that accepts only numbers between 20 and 30. This type of program is used for input validation.

```
/* Input validation between 20 to 30 */
main()
{
     int nInputval, nCorrectval;

     do
     {
          printf( “Enter a number between 20 and 30:“ );
          scanf ( “%d”, &nInputval );
     } while ( (nInputval<20) || (nInputval>30) );
     nCorrectval = nInputval;
     printf(“This is a valid number: %d”, nCorrectval);
}
```

5.2 Count-Controlled Loops

The *for statement* is considered count-controlled since the number of times it iterates is predetermined. It executes a statement (or a group of statements) for a specified and definite number of times. It keeps track of a counter whose value corresponds to the number of repetitions already performed. The repetition stops upon reaching the definite number of times specified. The syntax of the *for statement* is:

Format:

```
     for ( controlvar = initcount; expression; increment )
               statement;
```

where:

controlvar	is an arbitrary integer variable
initcount	is the starting count of the loop and the starting value of the *controlvar*
expression	is the condition for the loop to continue
increment	is the next value of the *controlvar*

To further understand the construct, let us have a very simple example:

```
for ( i = 1 ; i < = 100 ; i++ )
     statement;
```

This *for statement* tells us that statement will be executed 100 times. The *controlvar* i will have an initial value of 1 from the *initcount* followed by 2 then 3 and so on, as dictated by the statement in *increment*. The loop will stop when i is not less than or equal to 100.

It is important to remember that for the loop to be executed, the *expression* must evaluate to true. It is also important to note that in order for the loop to stop at one point, the statement in the *increment* must lead the *condition* to return false.

We are not restricted to increment the *for statement* by one (1). We can increment it by whatever value we want to.

```
     for ( i = 1; i <=100 ; i += 2 )
          statement;
```

This loop will no longer have 100 repetitions but 50. The reason is that the increment was changed to i = i +2. The value of the *controlvar* would be 1, 3, 5, 7, and so on until 99. When i is incremented by 2 again, the value would be 101 which turns the condition to false. Thus, the loop stops.

We can also count backwards if we want to. We simply have to change the increment into a decrement. Remember, the value of the condition must still be true.

```
     for ( i = 5 ;  i >= 1 ; i-- )
          statement;
```

This loop will have five repetitions. This time, the *controlvar* i will start with 5 going down to 1.

NOTE: Don't change the value of the control variable inside the loop body, otherwise, the result becomes unpredictable.

Example 1. Write a program that will get the sum of all integers from 1 to 1000.

```
/* sum of all integers 1 to 1000*/
main ()
{
     int nSum, I;

     nSum = 0;
     for ( i = 1; i <= 1000; i++)
          nSum = nSum + i ;
     printf("The sum of 1 to 1000 is %d.\n",nSum);
}
```

Example 2. Input a sequence of numbers and find their average. The number of values to be averaged must be input first.

```
/* get average of input */
main()
{    int nCount, nNumber, nSum, I ;
     double dAverage;

     printf( “Enter number of values to be averaged:” );
     scanf ( “%d”, &nCount );
     nSum = 0;
     printf ( “Enter the values:” );
     for ( I = 1; I <= nCount; I++ )
     {
          scanf ( “%d”, &nNumber );
          nSum = nSum + nNumber;
     }
     dAverage = (nSum + 0.0) / nCount;
     printf( “The ave. of the %d input values is %f \n”,
          nCount, dAverage );
}
```

Example 3. Write a program that will display a rectangle whose length and width are input by the user. For example, if the length is 5 and the width is 10, the output display would be:

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

```
/* make a square */
main()
{    int nLength, nWidth, I, J ;

     printf ( “Enter the length :” );
     scanf ( “%d”, &nLength );
     printf ( “Enter the width :” );
     scanf ( “%d”, &nWidth );

     for ( I = 1 ; I <= nWidth ; I++ )
          printf ( “*” );
     printf ( “\n” );
     for (I = 2; I <= nLength –1 ; I ++)
     {         printf ( “*” );
          for ( J = 2 ; J <= nWidth – 1; J++ )
                    printf ( “ “ );
               printf( “*\n” );
     }
```

```
          for ( I = 1 ; I <= nWidth ; I++ )
               printf ( "*" );
     }
```

NOTE: Placing a loop statement inside another loop statement is called a *nested loop*.

5.3 Infinite Loops

A loop that has an event that cannot be satisfied will continue to loop forever. Such loops are called infinite loops. Infinite loops are logical errors that may cause unpredictable consequences.

Consider the following program segment :

```
i = 0;
while (i < 10 )
{
     i--;
     printf ("Infinity \n");
}
```

**** Note: even without i--, this will result to an infinite loop*

The while statement is waiting for an event that will not happen. The control variable i is initially less than 10 and will continue to decrease. Since i will always be less than 10, the segment will never exit the loop.

5.4 Loop Conversions

Count-controlled loops can be converted to event-controlled loops. The for statement to while statement conversion is:

```
for (controlvar = initcount; condition; increment)
          statement;
```

becomes

```
controlvar = initcount;
while (condition)
{
     statement;
     increment;
}
```

5.4.1 Comparison of Loop Statements

Before selecting a looping structure, you must first model the loop using pseudo code. The choice will become obvious when the model is complete.

The *for statements* should be used when dealing with fixed numerical values, incremented or decremented as specified by the problem. While and do-while statements are ideal for loops with uncertain bounds (i.e. the duration of the loop can vary). In addition, control variables in while and do-while statements are not limited to simple ordinal values; while and do-while statements can use complex variables like characters and reals for control. However, while and do-while statements are more prone to errors since they are more flexible. Careless composition of while and do-while statements may lead to infinite loops.

A do-while statement can be converted into a while statement simply by copying the iterated statement in the do-while statement and replacing the do-while statement into a while statement.

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

becomes

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

It is necessary to copy the statement above the while statement since a do-while statement would execute the procedure at least once.

Converting the while statement into a do-while statement is not as simple. A while statement will prevent the entry into the loop if the condition is not valid, however, a do-while would allow at least one entry into the loop even if the condition does not hold.

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

becomes

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

Below is a program that computes the wages of employees. The program requires the number of hours the employee worked. The wage is computed at a fixed rate of 80 per hour.

```
#define HOURLY_RATE 80

/* compute wages */

main ()
{
     int nHoursWorked;
     double dWage;

     printf ( “Number of Hours Worked : ” );
     scanf ( “%d”, &nHoursWorked );
     dWage = nHoursWorked * HOURLY_RATE;
     printf ( “The salary is %.2f ”, dWage );
}
```

If the problem specified that the salary will be computed for 50 employees, we can use a for statement to iterate our program.

```
#define HOURLY_RATE 80

/* compute wages using for statement*/

main ()
{
   int nWorkers;
   int nHoursWorked;
   double dWage;
   for (nWorkers = 1; nWorkers <= 50; nWorkers++)
   {
          printf ( “Number of Hours Worked : ” );
          scanf ( “%d”, &nHoursWorked );
          dWage = nHoursWorked * HOURLY_RATE;
          printf ( “The salary is %.2f ”, dWage );
   }
}
```

This program would work for a fixed number of employees, specifically 50. However, if the company maintains an irregular number of employees, this will not be very practical. If only 40 employees need processing, there will be a surplus of 10 entries. If 60 employees need processing, the program would be executed twice leaving a surplus of 40 entries. This problem cannot be solved by the *for statement* and thus, needs to be addressed by another type of loop structure.

We may use a while statement to compute the wages.

```
#define HOURLY_RATE 80

/* compute wages using while statement */
main ()
{    char cWillCompute;
     int nHoursWorked;
     double dWage;

     printf ( “Number of Hours Worked : ” );
     scanf ( “%d”, &nHoursWorked );
     dWage = nHoursWorked * HOURLY_RATE;
     printf ( “The salary is %.2f ”, dWage );

     printf ( “Do you want to compute? (Y/N)” );
     scanf ( “%c”, &cWillCompute );

     while (cWillCompute == ‘y’ || cWillCompute == ‘Y’)
     {
          printf ( “Number of Hours Worked : ” );
          scanf ( “%d”, &nHoursWorked );
          dWage = nHoursWorked * HOURLY_RATE;
          printf ( “The salary is %.2f ”, dWage );

          printf ( “Do you want to compute? (Y/N)” );
          scanf ( “%c”, &cWillCompute );
     }
}
```

In this example, the program will continue to execute as long as the user enters ‘y’ or ‘Y’ to the question “Do you want to compute? (Y/N)”. Thus, the user may continue as long as he or she chooses.

However, the program executes several statements that are similar to the loop body before entering the loop. This repetition may be eliminated if we give a proper initial value to cWillCompute.

```
#define HOURLY_RATE 80

/* compute wages with initial value to cWillCompute */
main ()
{    char cWillCompute;
     int nHoursWorked;
     double dWage;

     cWillCompute = ‘Y’;
     while (cWillCompute == ‘y’ || cWillCompute == ‘Y’)
     {
          printf ( “Number of Hours Worked : ” );
          scanf ( “%d”, &nHoursWorked );
          dWage = nHoursWorked * HOURLY_RATE;
          printf ( “The salary is %.2f ”, dWage );

          printf ( “Do you want to compute? (Y/N)” );
          scanf ( “%c”, &cWillCompute );
     }
}
```

Or, we could use a do-while statement to solve the problem.

```
#define HOURLY_RATE 80

/* compute wages using do-while statement */
main()
{
     char cWillCompute;
     int nHoursWorked;
     double dWage;

     do
     {
          printf ( “Number of Hours Worked : ” );
          scanf ( “%d”, &nHoursWorked );
          dWage = nHoursWorked * HOURLY_RATE;
          printf ( “The salary is %.2f ”, dWage );

          printf ( “Do you want to compute? (Y/N)” );
          scanf ( “%c”, &cWillCompute );
     }while(cWillCompute == ‘y’ || cWillCompute == ‘Y’);
}
```

5.5 Nested Looping

Loops may be nested just like other control structures. Nested loops consist of an outer loop with one or more inner loops. Each time the outer loop is repeated, the inner loops are reentered, their loop control expressions are reevaluated, and all required iterations are performed.

***Example 1.* Loop statements for inner and outer loop are both count-controlled.**

```
     for ( i = 1; i < 4; i++)
     {
          printf (“Outer %d\n”, i);
          for ( j = 0; j < i; j++)
          {
               printf(“   Inner%d\n”, j);
          }
     }
```

***Example 2:* Loop statements for inner and outer loop are both event-controlled.**

```
#define HOURLY_RATE 80

main ()
{
   char cWillCompute;
   int nHoursWorked;
   double dWage;

   cWillCompute = ‘Y’;
   while (cWillCompute == ‘y’ || cWillCompute == ‘Y’)
   {
     printf ( “Number of Hours Worked : ” );
     scanf ( “%d”, &nHoursWorked );
     dWage = nHoursWorked * HOURLY_RATE;
     printf ( “The salary is %.2f ”, dWage );

     do
     {
          printf ( “Do you want to compute? (Y/N)” );
          scanf ( “%c”, &cWillCompute );
     }while(cWillCompute!= ‘y’ && cWillCompute != ‘Y’ &&
            cWillCompute != ‘n’ && cWillCompute != ‘N’);
   }
}
```

Example 3: **Combination of types of loop statement.**

```
j = 5;
do
{   for ( i = 1; i<j; i++ )
         printf ( “*” );
     printf ( “\n” );
     j--;
} while ( j > 0 );
```

Common Programming Errors

1. The three expressions in a for statement are separated by semicolons, not commas or other symbols. For this reason, it is wrong to write

```
for  ( expression1, expression2, expression3 )
                 statement1;
```

2. Do not place a semicolon at the end of the *for* (…) and the body of the for loop. The following is syntactically correct, but possibly logically erroneous if the user intends the *printf* statement to be inside the loop.

```
for  ( i = 0; i < 10; i++ );
    printf(“i = %d”, i);
```

3. The while statement does not include the word *do*. Thus, it is wrong to write

```
while ( expression ) do
      statement;
```

Self Evaluation Exercises

1. How many “*” will be printed on screen?

```
main()
{
     int x, y;

     x = -1;
     for (y = 4 ; y > x;  y--)
          printf (“*”);
}
```

2. How many characters will be printed on screen?

```
for (a = ‘a’; a <= ‘z’; a++)
     printf (“%c ”,a);
```

3. Write a program that will display the even numbers from 1 to 100.

4. Write a program that will compute for and display the sum of the numbers divisible by 3, ranging from 1 to 100.

5. Write a program that will compute and display the sum of the powers of 2 from the first power to the nth power, where n is a nonnegative integer given by the user.
   ex. if n = 4, then compute $2^1+2^2+2^3+2^4 = 30$; thus display 30
   ex. if n = 5, then compute $2^1+2^2+2^3+2^4+2^5 = 62$; thus display 62

6. Write a program that will compute for the following given n & x:
   $x^1/1 + x^2/2 + x^3/3 + ... + x^n/n$

7. Given an input n assumed one-digit, display a pattern.
   ex. if n = 5, display

```
    1
   1_2
  1_2_3
 1_2_3_4
1_2_3_4_5
```

8. Write a program that will display a pattern depending on the value of n entered by the user.
   ex. if n = 3, display

```
*   *
 * *
  *
```

   ex. if n= 4, display

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

9. Write a program to display all combinations of A and B (from 1 to 100) that will make the expression false: (5 * A) – (3 * B) > 30

10. Answer the questions that follow the code

```
scanf( “%d”, &s );
x = s;
y = s;
if (s >= 0)
     do
     {
          if (s > x)
               x = s;
          if (s < y)
               y= s;
          scanf ( “%d”, &s );
     }while  ( s >= 0 );
if (s < 0)
     printf( “%d%d”, x, y );
else printf( “none\n” );
```

   a. What does the program segment do?
   b. What value is being assigned to x? y?
   c. What value must be entered for s for the loop to stop?

11. Write a program that reverses an input number.

12. Write a program that gets as input a binary number and outputs its corresponding decimal equivalent. Declare your integer variables as long. A long declaration accommodates 8 digits for a number. Example:

    Enter a binary number: 1101
    Decimal equivalent: 13

Chapter Exercises

1. How many times will “Hello” be displayed?

```
main()
{    int a, i;

     a = 3;
     for (i = -1 ; i <= 4 – a; i++)
          printf (“Hello”);
}
```

2. Show the screen output of the given program segment.

```
m = 6; n = 0;
for (h = 1; h<= m; h++)
{
     z = 2 * h;
     n = n + z;
     printf (“H =%d, Z =%d, N =%d\n”,h,z,n);
}
```

3. How many characters will be printed on screen?

```
for ( b=’G’; b <= ‘Z’”; b++)
     printf (“%c ”,b);
```

4. Show the screen output of the given program segment.

```
for (i = 2 ; i <= 4; i++)
     for ( j = 5 ; j >= 3; j--)
          printf (“%d \n”, i+j);
```

5. Show the screen output of the given program segment.

```
for (i = 1; i<=5; i++)
{
     for (j=1; j<i ; j++)
               printf("*");
     printf("\n");
}
```

6. Given the value of n from the user, write a program that will display the first n even numbers. Ex. if n = 5, then display the first 5 even integers which are 2, 4, 6, 8, 10.

7. Write a program that accepts a number n and displays the sum of even numbers and the sum of odd numbers from 1 to n.

8. Write a program that will computer for n! which is the product of all numbers from 1 to n.

9. Write a program that will compute for a^x given real value a and positive integer x.

10. Write a program that reads in a number n and outputs the sum of the squares of the numbers from 1 to n.[SAVI89]

    ex. if n = 3, then display 14 because $1^2 + 2^2 + 3^2 = 14$.

11. Write a program that asks for the start month and end month and compute for the total number of days from the start month to the end month. Do not use if-then-else, use switch statement.

    ex. If start month = 2 and end month = 5, then
    Total = 28 + 31 + 30 + 31 = 120. Display 120.

12. Write a program that will display the following:

    4, 8, 12, 16, ..., 496

13. Write a program that reads in a number n, and then reads in n numbers. The numbers will be alternately added and subtracted.

    ex. If n = 5 and the numbers entered are 4, 14, 5, 6, 1,
    then compute for 4 + 14 - 5 + 6 - 1 = 18. Display 18.

14. Write a program that will ask the user to enter 10 numbers and display the largest number entered.

15. Write a program that will ask the user to enter 100 numbers and display on the screen the highest and lowest of these numbers.

16. Write a program that will ask the user for a number and display all the factors of the number.

17. Write a program that will ask the user for a number and display a message specifying whether the number is prime or composite.

18. Read any two numbers that represent an amount of money to be deposited and the annual interest rate. Compute and print the annual balances for the first 10 years.

19. Write a program that will ask the user for a number n and display the nth Fibonacci number F_n. F_n defined as follows:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_2 &= 0 + 1 = 1 \\ F_3 &= 1 + 1 = 2 \\ F_4 &= 1 + 2 = 3 \\ F_5 &= 2 + 3 = 5 \\ &\vdots \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

20. Construct a program to tell whether an input number is a palindrome. A palindrome is something that the same read backwards and forwards, such as 12321.[SAVI89]

21. Many banks and savings and loan institutions compute interest on a daily basis. On a balance of 1000 with an interest rate of 6%, the interest earned in one day is 0.06 multiplied by 1000 and then divided by 365, because it is only for one day of a 365 day year. This yields 0.16 in interest and so the resulting balance is 1000.16. The interest for the second day will be 0.06 multiplied by 1000.16 and then divided by 365. Design a program that takes three inputs: the amount of deposit, the interest rate, and a duration in weeks. The algorithm then calculates the account balance at the end of the duration specified.[SAVI89]

22. Write a program that would print out a part of the multiplication table. Get as input the start number and end number.

Example:

```
Enter start number: 3
Enter end number  : 5

Output is
   1    2    3    4    5    6    7    8    9    10
3  3    6    9    12   15   18   21   24   27   30
4  4    8    12   16   20   24   28   32   36   40
5  5    10   15   20   25   30   35   40   45   50
```

23. Write a program that will compute and display the sum of the factorials of the numbers from 1 to n, where n is a nonnegative integer given by the user.
    ex. if n = 3, then compute 1!+2!+3! = 9; thus display 9
    ex. if n = 4, then compute 1!+2!+3!+4! = 33; thus display 33

24. Write a program that will compute and display the sum of the powers of x from the first power to the nth power, where x and n are nonnegative integers given by the user.
    ex. if x = 3 and n = 4, then compute $3^1 + 3^2 + 3^3 + 3^4 = 120$; thus display 120
    ex. if x = 2 and n = 5, then compute $2^1 + 2^2 + 2^3 + 2^4 + 2^5 = 62$; thus display 62

25. Write a program that will display an n by n square of asterisks given n.
    ex. if n = 4, display

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

26. Write a program that will display a pattern depending on n.
    ex. if n = 4, display   ex. if n = 3, display

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

27. Given an input n assumed one-digit, display a pattern.
    ex. if n = 4, display

```
1
12
123
1234
```

28. Given an input n assumed one-digit, display a pattern.
    ex. if n = 4, display

```
1234
123
12
1
```

29. Write a program that will display a pattern depending on n.
    ex. if n = 4, display   ex. if n = 3, display

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

30. Given an input n assumed one-digit, display a pattern.
ex. if n = 4, display
```
   1
  21
 321
4321
```

31. Given an input n assumed one-digit, display a pattern.
ex. if n = 4, display
```
4 3 2 1
 4 3 2
  4 3
   4
```

32. Write a program that will create a rectangle based on length and width. Use 2 iterative/loop statements.
ex. if length = 5
width = 10

display
```
**********
*        *
*        *
*        *
**********
```

33. Write a program that will display a pattern depending on the value of n entered by the user.
ex. if n = 3, display
```
  *
 * *
* * *
 * *
  *
```
ex. if n= 4, display
```
   *
  * *
 * * *
* * * *
 * * *
  * *
   *
```

34. Write a program that will display a pattern depending on the value of n entered by the user.
ex. if n = 3, display
```
  *
 * *
*   *
```
ex. if n= 4, display
```
   *
  * *
 *   *
*     *
```

35. Write a program that will display a pattern depending on the value of n entered by the user.

ex. if n = 4, display

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

ex. if n= 5, display

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

36. Write a program that will compute for the following given n:
$1/2^1 + 2/2^2 + 3/2^3 + 4/2^4 + ... + n/2^n$

37. Write a program that will compute for the following given n & x:
$x^0/0! + x^1/1! + x^2/2! + x^3/3! + ... + x^n/n!$

38. Show the screen output of the following program.

```
/* strange program ??? */
main ()
{
     int num1, num2, x,y;

     num1 = 7; num2 = 4;
     for ( x = 1 ; x <= num2 ; x++)
     {
          for (y = 1 ; y <= num2-x; y++)
               printf(“%d ”,y);
          for (y = 1; y <= x; y++)
          {
               printf(“%d “,num1);
               num1 = (num1 + 1) % 10;
          }
          printf(“\n”);
     }
}
```

39. What will be the result given this program segment?

```
x = 1; y = 0;
while (x != 10)
{
     x = x + 2;
     y = y + x;
     printf(“%d%d\n”,x, y)
}
```

40. What does the following program segment do?

```
scanf (“%d”,&n);
i = 0;
while (i < n)
{
     j = (i + 1) * 2;
     printf(“%d\n”,j);
     i++;
}
```

41. Show the screen output of the following program.

```
main()
{  int    num,i,j,x,y,n,m,h;

   num = 341;
   n = 3;
   i = 0;

   while (i-1 <= n)
   {
     if (i == n+1)
          m = n-1;
     else m = n;

     if (i > 0)&& (i <= n)
     {    printf(“*”);
          h = num;
          for (j = 1; j<=n;j++)
          {    printf(“%d”,h %10);
               h = h / 10;
          }
          printf (“*\n”);
     }
     if (i != 0)
     {    x = num / 100;
          y = num % 100;
          num = y * 10 + x;
     }
     if ((i == 0) || (i == n+1))
     {    j = 0;
          do
            {  j = j + 1;
               printf(“*”);
            }while  (j != m+2);
          printf(“\n”);
     }
     i++;
   }
}
```

42. Write a program that will perform prime factorization.
    ex. Enter number: 24
        Prime factorization: 2, 2, 2, 3
    ex. Enter number: 7
        Prime factorization: 7

43. A perfect number is a positive integer that is equal to the sum of all those positive integers (excluding itself) that divide it evenly. The first perfect number is 6 because its divisors (excluding itself) are 1, 2, and 3, and because 6 = 1 + 2 + 3. Write a program to find the first three perfect numbers (include 6 as one of the three).[SAVI89]

44. Write a program that takes one real value as input and computes the first integer n such that 2^n is greater than or equal to the input value. The program should output both n and 2^n.[SAVI89]

45. Write a program to read in a real value x and output the integer n closest to the cube root of x. The value of x is assumed positive.[SAVI89]

46. Write a program that finds the lowest odd integer among the values entered by the user. Stop asking values when a value less than 1 has been entered.
    ex. if the values entered by the user are 3, 8, 1, 6, 3, 4, -5
        then display 1
    ex. if the values entered by the user are 6, 4, 8, 0
        then display 'No odd integer'

47. Write a program that will ask for a long number and count how many digits are there in the number.
    ex. Enter num: 10854
        Output: 5

48. Write a program that will ask for a long number and count how many digits in the number are even and how many are odd.
    ex. Enter num: 80572
        Output: 3 digits are even
                2 digits are odd

49. Write a program that will ask for a value for num and display the product of its even positioned digits (i.e., digits at the tens unit, thousands unit,...). Use long for num.
    ex. if num = 412473, then compute for 7*2*4=56; thus display 56
    ex. if num = 15678, then compute for 7*5=35; thus display 35
    ex. if num = 3, then display 0

50. Write a program that will ask for a value for num and display the product of its odd positioned digits (i.e., digits at the ones unit, hundreds unit,...). Use long for num.
    ex. if num = 12473, then compute for 3*4*1 = 12; thus display 12
    ex. if num = 5678, then compute for 8 * 6 = 48; thus display 48
    ex. if num = 3, then display 3

51. Write a program that asks for an input integer and displays the digits in reverse. Use long.

ex.	Enter number: 3562 Output: 2653	ex.	Enter number: 10240 Output: 04201

52. Write a program that accepts as input an integer in the range of 0 to 255 then outputs its corresponding 8-digit binary equivalent without leading 0's. Example,
    Enter decimal number: 19
    Equivalent binary #: 10011

53. Write a program that accepts as input an integer in the range of 0 to 255 then outputs its corresponding 8-digit binary equivalent with leading 0's. Example,
    Enter decimal number: 19
    Equivalent binary #: 00010011

54. Write a program that asks for a long value and swap every two digits in this value.
    ex. If n = 12345, display 21435
    If n = 811230, display 182103
    If n = 10101, display 01011

55. Write a program that asks for values from the user and find out which number was entered the most number of times (mode) and how many times this number was entered (frequency). Stop asking for values once a -1 has been entered. Assume that the numbers entered are positive and in increasing order and there is only one mode in the given inputs.
    ex. If the following numbers were entered:
    1, 2, 2, 2, 3, 3, 6, 6, 7, 7, 7, 7, -1
    The mode is 7 and its frequency is 4
    ex. If the following numbers were entered:
    2, 3, 3, 3, 4, 5, 5, 7, 8, 8, 8, 8, 8, -1
    The mode is 8 and its frequency is 5