Decision Making and Looping

Unit 5

INTRODUCTION

In the previous lessons we have learned about the programming structure, decision making procedure, how to write statements, as well as different types of decision or conditional statements and some basic control statement based program. In this unit we will learn about definition of loop, declaration and initialization of loop, various types of loop in C program and its applications and also how to handle loops in program. In C programming language loop is very useful statement.

Timeframe How long ?	We expect that this unit will take maximum 10 hours to complete.

Unit Structure

Lesson-1 Basic Concept On Loop Control Structures

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define programming loop.
- Understand the loop control structures.
- Explain uses of loop in program.

	Keywords	**Program, Loop, Control Structure, Entry, Exit**

INTRODUCTION TO LOOP

In practical, we may encounter many circumstances, when a block of code needs to be executed several number of times. We have already learned that in program, statements are executed sequentially. The first statement is executed first, followed by the second, third and so on. C language provides various control structures that allow for more complicated execution paths. In programming, a ***loop*** is a sequence or arrangement of instruction(s) that is continually repeated until a certain condition is reached. In looping, a sequence of statements are executed until some conditions for end of the ***loop*** are satisfied. It is a necessary idea that is commonly used in writing programs. A ***loop*** statement permits us to execute a statement or group of statements several times. A program loop contains two sections, such as *(1) body of the loop and (2) control statement.* The control statement tests definite conditions and then guides the repeated execution of the statements checked in the body of the loop.

Note it!

A loop is a sequence or arrangement of instruction(s) that is continually repeated until a certain condition is reached.

LOOP CONTROL STRUCTURES

A loop control structure may be classified either as the ***entry-controlled loop or as the exit-controlled loop.*** Both control structures are described in the following sections.

Entry-controlled loop

In the entry-controlled loop, the control conditions are tested before the start of the loop execution. It the conditions are not satisfied, then the body of the loop will not be executed. The flowchart of entry-controlled loop is shown in below:

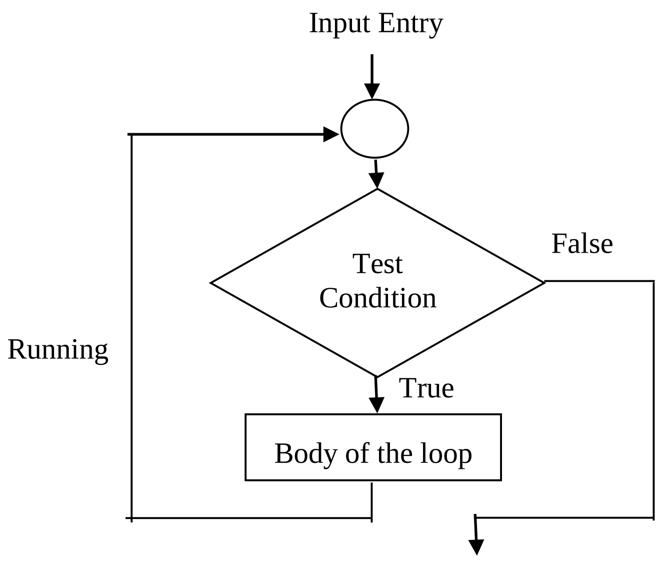


Figure 5.1.1: Entry-controlled loop structure

Exit-controlled loop

In the exit-controlled loop, the control conditions are tested at the end of the body of the loop. As result the body is executed unconditionally for the first time. The flowchart of exit-controlled loop is shown in below:

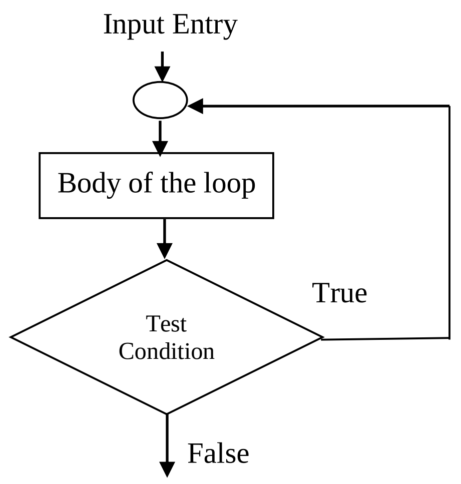


Figure 5.1.2: Exit-controlled loop structure

A loop control structure may be classified either as the entry-controlled loop or as the exit-controlled loop.

LOOPING PROCESS

Usually in a looping process, would include the subsequent four steps:

1. Setting and initialization of a counter variable.
2. Execution of the statements in the loop.
3. Test for a specified conditions for execution.
4. Increment or decrement the counter variable.

The programming C language provides three loops for performing loop operation, which are as follows: (i) while loop, (ii) do-while loop, and (iii) for loop. We will briefly describe above loops in the later lessons one by one.

Activity

1. Mention the significance of using loop in C language by your own concept.
   ……………………………..………………………………………………………………..
   ………………………………………………………………………………………………...
   …………………………...……………………………………………………………….

Summary Summary	
In this lesson	
▪ We have learned about C loop control structure.	
▪ Moreover we have learned how to design entry-controlled and exit-controlled loop structure flow chart.	

Assessment

Assessment

Fill in the blanks

1. A ***loop*** is a …………instruction(s).
2. A ***loop*** statement ……….. us to execute a statement or …………. several times.

Multiple Choice Questions (MCQ)

1. a ***loop*** is a sequence or arrangement of
   a) Variables b) Constants c) Instructions d) None of these
2. In looping, a sequence of statements are executed until-
   a) Some conditions for end of the loop are satisfied
   b) Only one conditions for end of the loop are satisfied
   c) Some variables for end of the loop are satisfied
   d) None of these
3. In the entry-controlled loop, the control conditions are tested
   a) Before the start of the loop execution
   b) After the start of the loop execution
   c) Middle of the loop execution
   d) End of the loop execution
4. In the exit-controlled loop, the control conditions are tested
   a) At the end of the body of the loop
   b) After the start of the loop execution
   c) Before the start of the loop execution
   d) None of these

Exercises

1. What is loop? Mention the major benefit of using loop in programming language.
2. Explain the loop control structures with flowchart.
3. Mention the looping process steps.

Lesson-2 Understanding While Loop

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define the *while* loop.
- Explain the *while* loop with syntax.
- Explain the *while* loop flowchart and example.

	Keywords	**Program, Loop, While loop, Syntax, Flowchart**

THE WHILE LOOP

We have already learned that loops are used in programming to repeat a specific block until some end condition is met. ***While*** loop is simple loop in C. A ***while*** loop in C programming repeatedly executes a target statement as long as a given condition is true. The basic form or syntax of while loop is as follows:

```
while( Test-condition)
{
   Body of the loop or statement(s);
}
```

Here, body of the loop or statement(s) may be a single statement or a block of statements. The ***Test-condition*** may be any expression, and true is any nonzero value. The ***while*** loop is an entry-controlled loop.The ***Test-condition*** is evaluated and if the condition is true then the body of the loop or statements

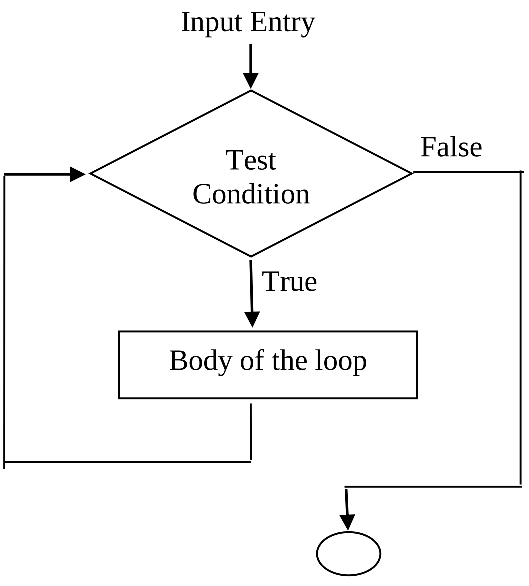


Figure 5.2.1: Flowchart of while loop

are executed. After execution of the body of the loop, the ***Test-condition*** is once again evaluated and it is true, the body of the loop is executed again. This process is repeated until the ***Test-condition*** finally becomes false and the programs control is transferred out of the loop. Finally we can say that, in while loop Test-condition is tested before the body of the loop is executed. The flowchart of while loop is shown in figure 5.2.1. Here, the important point is noted that a ***while*** loop might not execute at all, when the ***Test-condition*** is tested and the result is false, body of the loop will be skipped and the next statement after the while loop will be executed.

Note it!

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

Program 5.2.1 write a C program that prints 10 to 15 using while loop

```
#include <stdio.h>
#include <conio.h>
void main ()
 {
      int a = 10;
     while( a <16 )
         {
            printf("value of a: %d\n", a);
            a++;
         }
    getch();
}
```

………………………………………………………………………………….

Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Program 5.2.2 write a C program to find out factorial value of a given integer number using while loop

```
#include <stdio.h>
void main()
{
    int number;
    long int factorial;
    printf("Enter an integer Number: ");
    scanf("%d",&number);
    factorial = 1;
    while (number > 0)
    {
        factorial = factorial * number;
        --number;
    }
    printf("Factorial value is = %ld", factorial);
}
………………………………………………………………………………….……….
Output
Enter an integer Number: 6
Factorial value is = 720
Enter an integer Number: 10
Factorial value is = 3628800
```

Program 5.2.3 write a C program that prints only odd numbers from 1 to 30 using while loop

```
#include <stdio.h>
#include<conio.h>
void main ()
{
     int a = 1;
     printf("Odd Numbers Till 30 are: \n");
     while( a < 30 )
     {
        printf("%d \t", a);
        a+=2;
      }
   getch();
}
……………….……………………………………………………………..
Output
1   3    5    7    9    11    13   15    17    19    21     23     25    27     29
```

Note it!

A while loop might not execute at all, when the Test-condition is tested and the result is false, body of the loop will be skipped and the next statement after the while loop will be executed.

Activity

1. What will be the output of the following code segment

```
#include <stdio.h>
#include <conio.h>
void main ()
 {
     int p = 5;
     while( p >20 ) {
            printf("value of a: %d\n", p);  p++;
         }
     printf(" Hello Good day!!");
}
```

2. Find out errors from the following code segment

```
void main ()
 {
      int p = 5;
     while  p =5 ;
         {
            printf("%d\n", a); p;
         }printf(" Hello Good day!!");
  }
```

Summary Summary	
In this lesson ▪ We have learned about C while loop with its syntax. ▪ We have also learned how a while loop works in program.	

ASSIGNMENT

Assignment

1. Write a C program that find out the odd and even numbers from N integer numbers using while loop.
   …………………………………………………………………………………
   ……………………..………………………………………………………………..
2. Write a C program to print the result of the equation y^x using while loop.
   …………………………………………………………………………………
   ………………………………………………………………………………….

Assessment

Assessment

Write "T" for true and "F" for false the following sentences:

1. A *while* loop in C programming repeatedly executes a target statement as long as a given condition is false.
2. In *while* loop Test-condition finally becomes true and the programs control is transferred out of the loop.

Multiple Choice Questions (MCQ)

1. In while loop body of the loop or statement(s) may be
   a) A single statement or a block of statements b) Only single statements
   c) Only two blocks d) None of these
2. A *while* loop in C programming repeatedly executes a target statement as long as a given
   a) Condition is true.
   b) Condition is false
   c) Both true and false is possible
   d) None of these
3. In while loop when the *Test-condition* is tested and the result is false, body of the loop will be-
   a) Printed b) Terminated c) Executed d) Skipped

Exercises

1. What is while loop? Mention the major benefit of using while loop in programming language.
2. Explain the while loop control structures with flowchart.
3. Write a C program to calculates the power value of equation $P=2^N$ using while loop.
4. Write a C program to find out the value of following series using while loop:
   $Y= 1^1+2^3+3^5+$………………………………….N^n

Lesson-3 Understanding Do-While Loop

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define *do-while* loop.
- Explain the *do-while* loop with syntax.
- Explain the *do-while* loop flowchart and example.

	Keywords	**Program, Loop, Do-while loop, Syntax, Flowchart**

THE DO-WHILE LOOP

We have learned that, in while loop a Test-condition is tested before body of the loop is executed. Consequently, if the Test condition is not satisfied at the first time, then the body of the loop may not be executed at all. But in some situations it might be necessary to execute the body of the loop or statement(s) before the test is executed. Such circumstances can be handled with the help of ***do-while*** loop. The general syntax of ***do-while*** loop is as follows:

```
do
  {
     Body of the loop or statement(s);

  }while(Test-condition);
```

In ***do-while*** loop, the program proceeds to evaluate the body of the loop or statement(s) first. In this loop, the ***Test-condition*** in while statement is evaluated at the end of the loop.

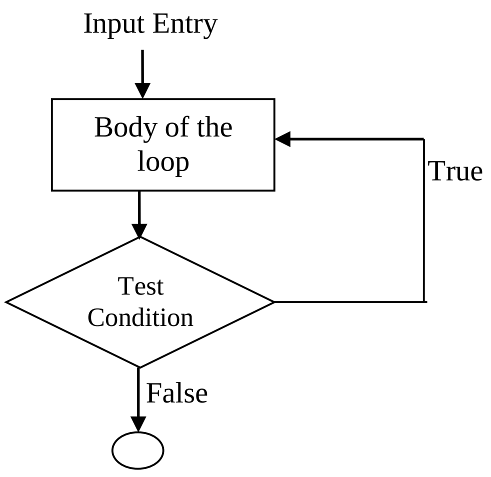


Figure 5.3.1: Flowchart of do-while loop

If the ***Test-condition*** is true, the program continues to evaluate the body of the loop once again. This process continues as along as the ***Test-condition*** is true. If the ***Test-condition*** is false, the loop will be terminated and program control goes to the statement that appears after while statement A ***do...while*** loop is similar to a while loop, but the point that it is exit-controlled loop and therefore the body of the loop is always executed at least one time. The flowchart of *do-while* loop is shown in figure 5.3.1.

Note it!

In do-while loop, the program proceeds to evaluate the body of the loop or statement(s) first. In this loop, the Test-condition in while statement is evaluated at the end of the loop

Program 5.3.1 Write a program to calculate the average of N numbers and display the result using do-while loop.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
   int number, count =1;
   float x, average, sum = 0.0;
   printf("How Many Numbers?:");
   scanf("%d",&number);
   do
    {
      printf(" Enter the %d Number = ",count);
      scanf("%f",&x);
      sum = sum + x;
      ++count;
    }while(count < = number);
   average = sum/number;
   printf("\n The average is: %f",average);
 getch();
}
…………………………………………………………………………….
Output
How Many Numbers?: 3
Enter the 1 Number =10
Enter the 2 Number =20
Enter the 3 Number =30
The average is: 20.000000
```

Program 5.3.2 Write a C program that convert a line of lowercase text to Uppercase text using do-while loop.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#define EOL '\n'
void main ()
{
   char letter[80];
   int value,count=-1;
   printf("Type text or line as lowercase:\n");
   do
   {
      ++count;
   }while((letter[count] = getchar())!= EOL);
   value = count;
   count=0;
   do
   {
      putchar(toupper(letter[count]));
      ++count;
   }while(count < value);
 getch();
}
………………………………………………………………………………
Output
Type text or line as lowercase:
i am a man.  i read in bangladesh open university
I AM A MAN. I READ IN BANGLADESH OPEN UNIVERSITY
```

Program 5.3.3 Write a C program that print the multiplication table of 5 from 1 to 10 using do-while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int value=1;
    printf("Multiplication Table of 5 from 1 to 10:\n");
    do
    {
       printf("5 * %d = %d\n",value,5*value);
       value++;
    }while(value<=10);
  getch();
}
……………………………………………………………………………….
Output
Multiplication Table of 5 from 1 to 10:
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Activity

1. What will be the output of the following code segment:

```
#include<stdio.h>
void main()
{
   int value=2;
```

```
        do
        {
           printf("6 * %d = %d\n",value,6*value);
           value++;
        }while(value <= 6);
     }
```

Summary Summary	
In this lesson ▪ We have learned about C do-while loop with its syntax. ▪ We have also learned how a do-while loop works in program.	

ASSIGNMENT

Assignment

1. Write a C program that display factorial value of n number using do-while loop.
   ……………………………………………………………………………………..
   ……………………………………………………………………………………..

Assessment

Assessment

Write "T" for true and "F" for false the following sentences:

1. In do-while loop, the program proceeds to evaluate the body of the loop or statement(s) first.
2. If the Test-condition is false in do while loop, the program continues to evaluate the body of the loop once again.

Exercises

1. What is do while loop? Mention the major benefit of using do while loop in programming language.
2. Explain the do while loop control structures with flowchart.

Lesson-4 Understanding For Loop

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define the *for* loop.
- Explain the *for* loop with syntax.
- Explain the *for* loop flowchart and example.

	Keywords	**Program, Loop, For Loop, Syntax, Flowchart**

FOR LOOP

We have learned in previous lesson about while and do-while loops. The ***for*** loop is the third and most commonly used loop in C program. It is another entry-controlled loop that delivers shorter loop control structure. The ***for*** loop is a recurrence control structure that allows us to proficiently write a loop that needs to perform an exact number of times.

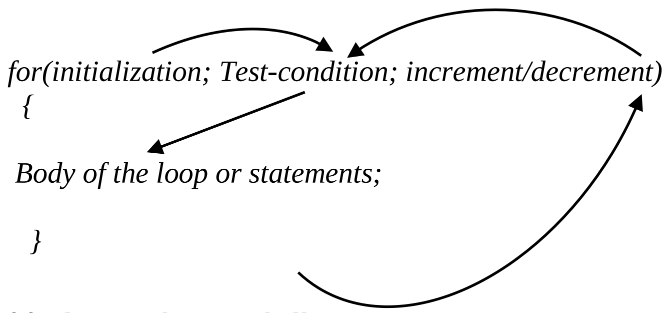
The general form or syntax of ***for*** loop is as follows:

```
for(initialization; Test-condition; increment/decrement)
 {
    Body of the loop or statements;
  }
```

The execution procedure of ***for*** loop is described below:

1. In loop, the ***initialization*** step is performed first, and only once. This stage allows us to announce or declare and initialize any loop control variables. We are not necessary to put a statement here, as long as a semicolon appears. Initialization of control variable using assignment operator such as *i=1, count=0, value=1* etc. These initialization variables are known as loop control variable.
2. Next, the ***Test-condition*** is evaluated. ***Test-condition*** is a relational expression such as count <=10 or I >20 etc. that determines when the loop will exit. If ***Test-condition*** is true, then the body of the loop is executed. If ***Test-condition*** is false, the body of the loop does not execute and the flow of control jumps to the next statement just after ***for*** loop.
3. After execution of the body of the ***for*** loop, the flow of control jumps back up to th*e* ***increment/decrement*** statement. This statement allows us to update (i.e., increment or decrement) any loop control variable.
4. After control variable updated (i.e., increment or decrement), the ***Test-condition*** is now evaluated again. If ***Test-condition*** is true, the loop executes and the process repeats itself and if the ***Test-condition*** becomes false, the ***for*** loop terminates.

The execution procedure is shown clearly as follows



```
for(initialization; Test-condition; increment/decrement)
{

Body of the loop or statements;

}
```

The flowchart of for loop is shown in bellow:

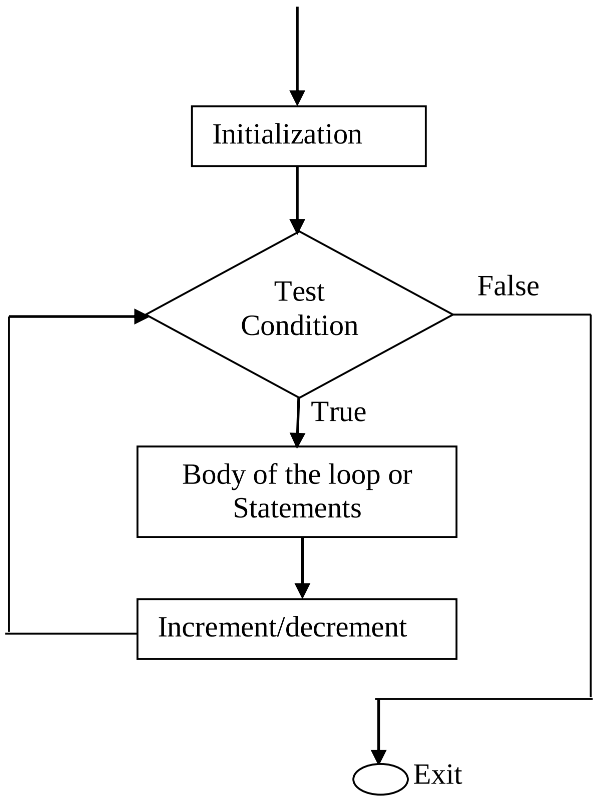


Figure 5.4.1: Flowchart of for loop

Note it!

The for loop is a recurrence control structure that allows us to proficiently write a loop that needs to perform an exact number of times

Program 5.4.1 Write a C program that display 1 to 10 using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i;
```

```
    for(i=1;i<=10;i= i+1)
    {
       printf("%d\t",i);
    }
  getch();
}
………………………………………………………………………………..
Output
1    2    3    4    5    6    7    8    9    10
```

Program 5.4.2 Write a C program that calculate the value of the following series using for loop Y=1+3+5+7+……………………………………………………………………N.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i, N, sum=0;
    printf("Enter the value N of series: ");
    scanf("%d",&N);
    printf("\nThe Total value of series is: ");
    for(i=1;i<=N; i=i+2)
    {
       sum=sum+i;
    }
   printf("%d",sum );
  getch();
}
…………………………………………………………………………………
Output
Enter the value N of series: 10
The Total value of series is: 25
```

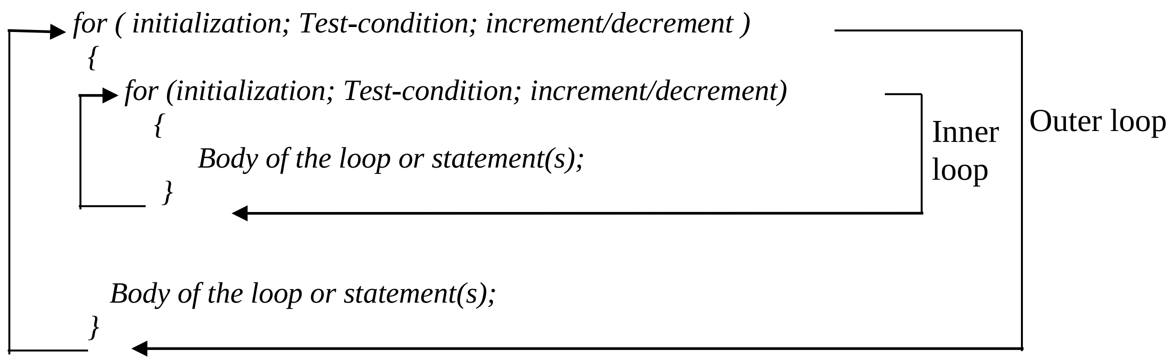
Program 5.4.3 Write a C program that calculate the value of the following series using for loop $Y=1^2+3^2+5^2+7^2+$……………………………………………………………………N^2.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    long int i, N, sum=0;
    printf("Enter the value N of series: "); scanf("%ld",&N);
    printf("\nThe Total value of series is: ");
    for(i =1;i<=N; i=i+2)
    {
       sum=sum+pow(i,2);
    }
   printf("%ld",sum ); getch();
}
………………………………………………………………………………..
Output
```

```
Enter the value N of series: 20
The Total value of series is: 1330
```

NESTING OF FOR LOOP

C programming allows us to use one loop inside another loop. Nesting of loops, that is one ***for*** loop within another ***for*** loop statement. This process is called nesting of ***for*** loop. Tow loops can be nested as follows:



```
for ( initialization; Test-condition; increment/decrement )
 {
   for (initialization; Test-condition; increment/decrement)
     {
        Body of the loop or statement(s);
     }


   Body of the loop or statement(s);
 }
```

Nesting of loops, that is one for loop within another for loop statement. This process is called nesting of for loop

Program 5.4.4 Write a C program that print the following pattern using nested for loop.

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6

```
#include<stdio.h>#include<conio.h>#include<math.h>
void main()
{
   int row,col;
   for(row=1;row<=6;row++)
   {
      for(col=1;col<=row;col++)
      {
         printf("%d ", col);
      }
      printf("\n");
   }getch();
}
……………………………………………………………………………
```

Output

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

```
1 2 3 4 5 6
```

Program 5.4.5 Write a C program that print the following pattern using nested for loop.

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

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
   int row,col,j,space=5;
   for(row=0;row<5;row++)
   {
      for(col=0;col<space;col++)
      {
      printf(" ");
      }
      for(j=0;j<=row;j++)
      {
         printf("* ");
      }
      printf("\n");
      space--;
   }
  getch();
}
```

…………………………………………………………………………………

Output

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

Program 5.4.6 Write a C program that print all prime numbers from 1 to N range using nested for loop.

```
#include <stdio.h>
void main ()
{
   int i, j,N;
   printf("Enter the value of range N: ");
   scanf("%d",&N);
   for(i = 1; i<N; i++)
     {
         for(j = 2; j <= (i/j); j++)
             if(!(i%j)) break;
             if(j > (i/j))
                 printf("%d is prime\n", i);
   }
}
…………………………………………………………………………..
```

Output

```
Enter the value of range N: 20
1 is prime
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
```

Activity

1. Analyze each of the program segments that follow and determine how many times the body of each loop will be executed:
   i.
   ```
   int i;
   for( i=0;i<=8; i=i+2/5)
   {
       ……………………
   }
   ```
   ii.
   ```
   int m=10, n=7;
   while(m%n>=0)
   {  …………………..
     m=m+1;
     n=n+2;
   }
   ```
2. Find errors, if any in each of the following looping segments:
   i.
   ```
   for(x=1,x>10;x=1)
   {
       printf(" %d test");
     ………………………..
   }
   ```
   ii.
   ```
   for (p=10;p>0;)
   p=p-1;
   printf("%f",&p);
   ```

Summary Summary	
In this lesson ▪ We have learned about C for loop with its syntax. ▪ We have also learned how a for loop works in program.	

ASSIGNMENT

Assignment

1. Write a C program that display factorial value of n number using for loop.
   ……………………………………………………………………………..
   ……………………………………………………………………………..
2. Write a program that print the result of the following series:
   Y= $1+x+x^2+x^3+x^5+$……………………………………..x^n

Assessment

Assessment

Write "T" for true and "F" for false the following sentences:

1. In for loop, the initialization step is performed first, and infinite.
2. In for loop Test-condition is an assignment expression.
3. The for loop is a recurrence control structure.
4. In for loop if Test-condition is false, then the body of the loop is executed.

Multiple Choice Questions (MCQ)

1. In for loop initialization of control variable using-

   a) Assignment operator b) Relational operator c) Logical operator d) None of these

2. For loop is another-

   a) Entry-controlled loop that delivers shorter loop control structure.
   b) Exit- controlled loop that delivers shorter loop control structure.
   c) Exit- controlled loop that delivers longer loop control structure.
   d) None of these

3. After execution of the body of the ***for*** loop, the flow of control-

   a) Jumps back up to the *increment/decrement* statement.
   b) Jumps down up to the *increment/decrement* statement.
   c) Jumps back up to the *assignment* statement.
   d) Jumps back up to the *test-control* statement.

Exercises

1. What if for loop? Discus with its syntax
2. Distinguish among while, do-while and for loop with an example
3. Describe for loop flowchart with an example.
4. Write a C program that prints the following pattern using for loop

```
A
A A
A  A A
A  A  A A
A  A  A  A A
```

5. Write a C program that prints the following pattern using for loop:

```
      1
    2   3
  3   4   5
4   5   6   7
```

6. Write a C program that prints the following pattern using for loop

```
       1
     2   2
   3   3   3
 4   4   4   4
```

7. Write a C program to evaluate the following investment equation:

$$V= P(1+r)^n$$

Lesson-5 Understanding Break And Continue Statements

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define break statement.
- Define continue statement.
- Explain uses of break and continue statements in program.

✓	**Keywords**	**Program, Break, Continue, Statement, Exit, Flowchart**

JUMPING PROCESS IN LOOPS

We have already known loops are used to perform a set of operations repeatedly until the control variable fails to satisfy the test condition. The number of periods a loop is repeated is decided in advance and the test condition is written to accomplish this goal. In most cases, when performing a loop it becomes necessary to skip a part of the loop or leave the loop as soon as a certain condition happens. Programming C permits jump from one statement to another within a loop as well as jump out of the loop. So it is necessary to skip some statements inside the loop or terminate the loop immediately without checking the test condition. In such cases, ***break*** and ***continue*** statements are used. These statements are described below with program.

BREAK STATEMENT

The ***break*** is a built-in keyword in C. The ***break*** statement allows us to departure a loop from any point within its body, avoiding its regular termination process or expression. In practical, when the ***break*** statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop. We have seen the use of ***break*** statement in switch statement. The general format of break statement is simply:

break;

When loops are nested, the ***break*** would only exit from the loop containing it. This means that break will exit only a single loop. The break statement can also be used in *while, do-while,* and *for* loops. The use of break statement procedure is illustrated in figure 5.5.1.

```
while(Test-condition)
 {
       ……………
       ……………
       if(condition)
          break;
       …………
 }
……………………
```

(a) break in while loop

```
do
  {
     ………………
     if(condition)
          break;
       ……………….
       ……………….
  }while(Test-condition);
…………………………
```

(b) break in do-while

```
for(init, Test-condition; increment)
 {
    ………………..
    ……………….
     if(condition)
        break;
     ……………
     ……………
 }
………………………
```

(c) break in for loop

```
for((init, Test-condition; increment)
{
    …………………………
    …………………………
    for(init, Test-condition; increment)
      {
          ………………..
          ……………….
           if(condition)
              break;
           ……………
           ……………
       }
    ………………..
}
```

(d) break in nested for loop

Figure 5.5.1 break used in while, do-while and for loops

The flowchart of break statement is shown in below:

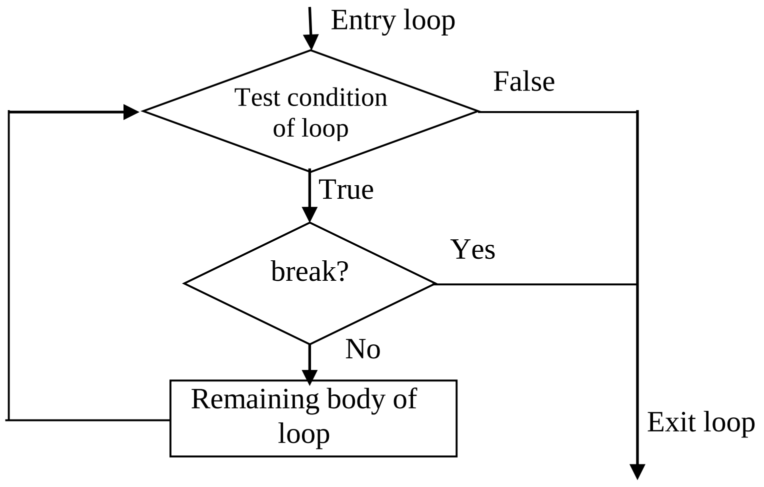


Figure 5.5.2: Flowchart of break statement

Note it!

When the break statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop

Program 5.5.1 Write a C program that calculates sum and average until user enters positive number. When enter negative number program will be terminated.

```
# include <stdio.h>
#include<conio.h>
void main()
{
   int i;
   float number, sum = 0.0, average=0.0;
```

```
    printf("The program calculates maximum 10 Numbers.\n");
    printf("The program will terminate, if enter negative number.\n");
    for(i = 1; i <= 10; ++i)
    {
        printf("Enter a n%d: ",i);
        scanf("%f",&number);
        /* If user enters negative number, loop is terminated*/
        if(number < 0.0)

            break;
        sum + = number;
    }
    average = sum/(i-1);
    printf("Total Sum = %.2f\n",sum);
    printf("Average is = %.2f",average);
getch();
}
……………………………………………………………………………......
```

Output

```
The program calculates maximum 10 Numbers.
The program will terminate, if enter negative number.
Enter a n1: 10
Enter a n2: 20
Enter a n3: 30
Enter a n4: -3
Total Sum = 60.00
Average is = 20.00
```

Program 5.5.2 Write a C program that takes numbers from user until enter 15.

```
# include <stdio.h>
#include<conio.h>
void main()
{
    int number ;
    printf("Enter numbers one by one :");
  for ( ; ; )
    {
      scanf("%d" , &number) ;
        if ( number = =15 )
           break ;
    }
printf("Your last Entered number is: %d \n",number);
printf("End of an infinite loop...\n");
getch();
}
…………………………………………………………………………………
```

Output

```
Enter numbers one by one :10
100
456
789
15
Your last Entered number is: 15
```

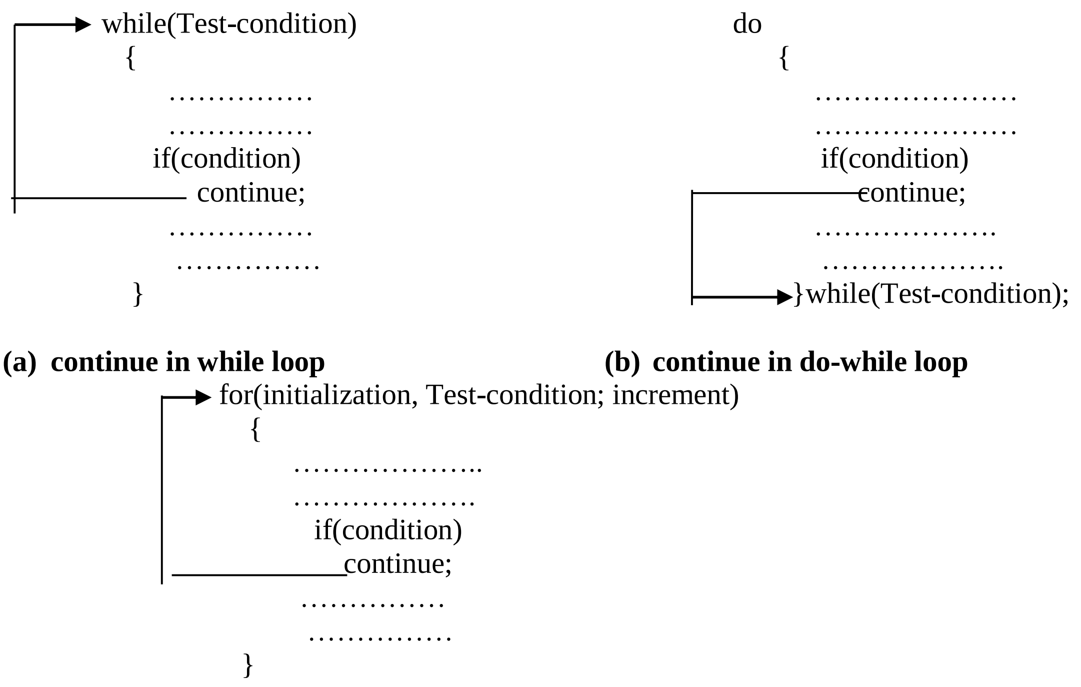
```
So End of an infinite loop...
```

CONTINUE STATEMENT

The ***continue*** is a built-in keyword in C. Throughout the loop operations, it may be essential to skip a part of the body of the loop under certain conditions. Like ***break*** statement C supports another statement called the ***continue*** statement. The ***continue*** statement is used to avoid the remainder of the current pass through a loop. However, when a ***continue*** statement is encountered, then the loop does not terminate. This means that, the ***continue*** statement causes the loop to be continued with the next iteration after skipping any statements in between. Actually, ***continue*** statement tells the compiler "Skip the following statements and continue with the next iteration." The general format of ***continue*** statement is:

continue;

The continue statement is also used in different types of loops like *while, do-while* and *for* loops. The use of continue statement in different loops are illustrated in following 5.5.3 figure. The continue process is shown with an arrow line in figure 5.5.3.



```
while(Test-condition)
 {
     ..............
     ..............
   if(condition)
       continue;
     ..............
     ..............
 }
```

(a) continue in while loop

```
do
 {
     ....................
     ....................
   if(condition)
       continue;
     ..................
     ..................
 }while(Test-condition);
```

(b) continue in do-while loop

```
for(initialization, Test-condition; increment)
 {
     ...................
     ..................
   if(condition)
   continue;
     ..............
     ..............
 }
```

(c) continue in for loop

Figure 5.5.3: Uses of continue in different types of loops

In ***while*** and ***do-while*** loops, ***continue*** causes the control to go directly to the ***Test-condition*** and then to continue the iteration process, on the other hand in ***for*** loop, the increment/decrement section of the loop is performed before the ***Test-condition*** is assessed.

The flowchart of ***continue*** statement is shown in below:

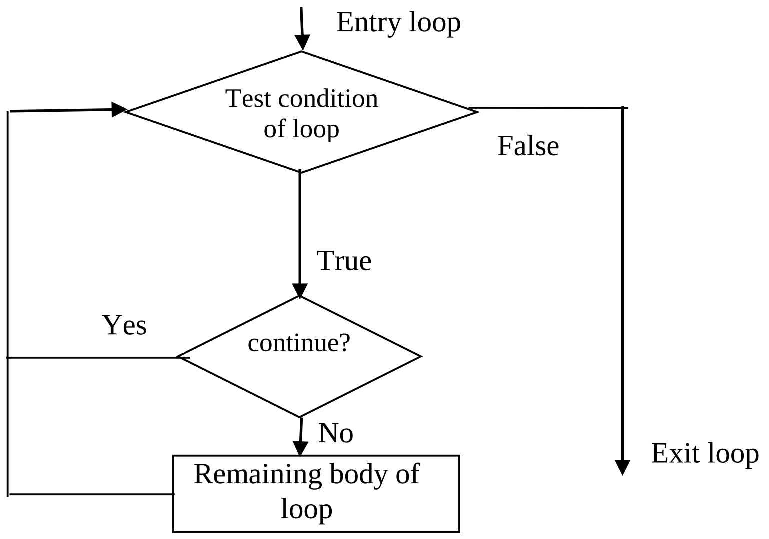


Figure 5.5.4 Flowchart of continue statement

Note it!

continue statement tells the compiler "Skip the following statements and continue with the next iteration"

Program 5.5.3 Write a C program that calculate sum of maximum 10 numbers. Negative numbers are skipped from calculation.

```
#include <stdio.h>
#include<conio.h>
void main()
{
   int i;
   float number, sum = 0.0;
   printf("Enter 10 Numbers.\n");
   printf("Negative number is skipped from calculation.\n");
   for(i=1; i <= 10; ++i)
   {
      printf("Enter a number n%d: ",i);
      scanf("%f",&number);
      /*If user enters negative number, loop is terminated*/
      if(number < 0.0)
      {
         continue;
      }
      sum += number;
   }
   printf("Total Sum = %.2f",sum);

  getch();
}
……………………………………………………………………………………
```

Output
```
Enter 10 Numbers.
Negative number is skipped from calculation.
Enter a number n1: 10
Enter a number n2: 20
Enter a number n3: -90
Enter a number n4: 40
Enter a number n5: 50
Enter a number n6: 46
Enter a number n7: 78
Enter a number n8: 90
Enter a number n9: 34
Enter a number n10: 23
Total Sum = 391.00
```

Activity

1. what will be the output of the following program segment:
```
#include <stdio.h>
#include<conio.h>
void main()
{
    int i=0,x=0;
    for(i=1;i<10;++i)
    {

        if(i%2==1)
          x+=i;
         else
            x--;
         printf("%d ",x);
         break;
    }
    printf("\nx=%d",x);
}
```
2. what will be the output of the following program segment:

```
#include <stdio.h>
#include<conio.h>
void main()
{
    int i = 0,j,x = 0;
    for(i = 0;i<5;++i)
    {
    for(j = 0;j<i;++j)
          x += (i + j - 1);
         printf("%d  ",x);
         break;
    }
    printf("\nx = %d",x);
}
```
3. what will be the output of the following program segment:
```
void main()
{
    int i=0,x=0;
    for(i=1;i<10;++i)
    {
       if(i%2 = =1)
         x+=i;
        else
           x--;
        printf("%d  ",x);
        continue;
    }
    printf("\nx=%d",x);
}
```

Summary Summary	
In this lesson ▪ We have learned about break statement with its syntax. ▪ We have also learned about continue statement with its syntax.	

ASSIGNMENT

Assignment

1. What will be the output of the following program

```
#include <stdio.h>
#include<conio.h>
void main(){
    int i,j,k,x=0;
    for(i=0;i<5;++i){
    for(j=0;j<i;++j){
      switch(i+j-1){
        case -1:
        case 0:
            x+=1; break;
        case 1:
        case 2:
        case 3:
            x+=2;
        default:
            x+=3;
        } printf("%d",x);
    } printf("\nx=%d",x);}
 }
```

Assessment

Assessment

Write "T" for true and "F" for false the following sentences:

1. C permits jump from one statement to another within a loop as well as jump out of the loop.
2. The break statement allows us to exit a loop from any point within its body.
3. When the break statement is encountered inside a loop, the loop is immediately terminated.

4. In while and do-while loops, continue causes the control to go directly to the Test-condition.

Multiple Choice Questions (MCQ)

1. When loops are nested, the break would only-
   a) Exit from the loop containing it
   b) Continue from the loop containing it
   c) Nested from the loop containing it
   d) None of these
2. The break statement allows us to departure a loop from any point within its body-
   a) avoiding its regular termination process or expression
   b) permitting its regular termination process or expression
   c) exit from the loop containing it
   d) continuing its regular termination process or expression
3. The continue statement tells the compiler-
   a) Continue the following statements and skip with the next iteration
   b) Skip the following statements and continue with the next iteration
   c) Test the following statements and continue with the next iteration
   d) None of these

Exercises

1. Why are break and continue statements used in program?
2. Mention the benefits of break statement with its flowchart.
3. Explain the uses of break statement in different types of loops.
4. Explain the uses of continue statement in different types of loops.
5. Describe distinguish between break and continue statement with an example.

Lesson-6 Understanding goto Statement

Learning Outcomes

Outcomes

Upon completion of this lesson you will be able to

- Define goto statement with syntax.
- Explain the uses of goto statement.

✓	**Keywords**	**Program, goto, Statement, Exit, Flowchart**

GOTO STATEMENT

In the previous lessons, we have already learned the uses of different types of statements those are used to directly exit or continue the execution of statements in different types of loops. Another statement that is used in C program to transfer the control one place to another place of a program is known as ***goto*** statement.

The ***goto*** statement is used to change the normal sequence of C program execution by transferring control to any place of the program. Since, the ***goto*** statement can transfer the program control to any place in a program, so it is convenient to provide branching within a loop.

This statement is also used to exit from deeply nested loops when an error occurs. Actually, ***goto*** statement provides an unconditional jump from the ***'goto'*** to a labeled statement in the same function. The general form or syntax of ***goto*** statement is:

Target statement

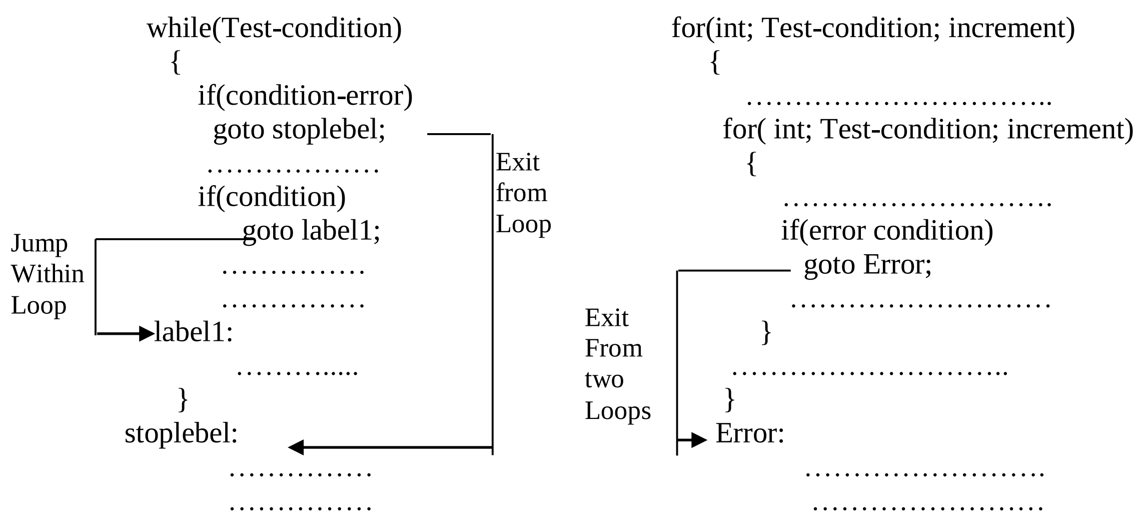
goto   Label_name;

Here, ***Label_name*** is an identifier that is used to label the target statement to which control will be transferred. The ***Label_name*** can be any ordinary text except C keyword and it can be set anyplace in the C program.

An important note that, target statement must be labeled and ***Label_name*** must be followed by a colon (:). Therefore the target statement will seem as:

Label_name:
Statement(s);

Every labeled statement within the program must have a unique label, which is no two statements can have the same label. Jumping within and exiting from the loops with ***goto*** statement is shown in following figure 5.6.1.



```
while(Test-condition)
  {
     if(condition-error)
       goto stoplebel;
     ………………
     if(condition)
          goto label1;
       ……………
       ……………
  label1:
        ………......
   }
stoplebel:
       ……………
       ……………
```

```
for(int; Test-condition; increment)
   {
     ……………………………..
   for( int; Test-condition; increment)
     {
        ……………………….
        if(error condition)
          goto Error;
        ………………………
     }
   ………………………..
   }
 Error:
        ………………………
        ……………………
```

Figure 5.6.1 Jumping within and exiting from the loops with goto statement

The flowchart of ***goto*** statement is shown in below:

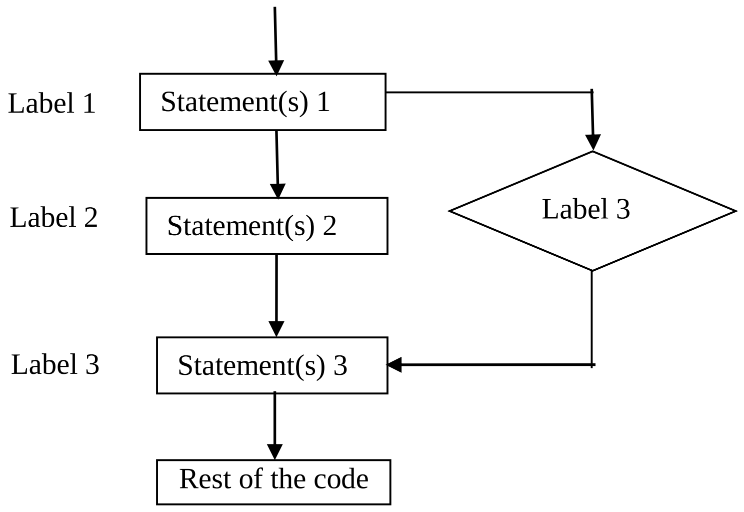


Figure 5.6.2 flowchart of goto statement

Note it!

The goto statement is used to change the normal sequence of C program execution by transferring control to any place of the program

Note it!

The use of goto statement is extremely discouraged in any programming language because it makes tough to trace the control flow of a program, making the program hard to understand and rigid to change. Any program that uses a goto can be redrafted to avoid them

Some ***goto*** programming examples are shown in below:

Program 5.6.1 write a C program that evaluate the following series using goto statement $\frac{1}{1-x} = 1 + x + x^2 + x^3 + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + x^n$ **for -1< x <1 to evaluate to 0.01 percent accuracy. Goto statement is used to exit the loop.**

```
#include <stdio.h>
#include<conio.h>
#define Loop 100
#define accuracy 0.0001
void main()
{
   int n;
   float x,term,sum = 0.0;
   printf("Enter the value of x: ");
   scanf("%f",&x);
   for(term =1,n = 1;n <= Loop; ++n)
   {
      sum += term;
      if(term < = accuracy)
         goto output; /* Exit from loop*/
      term*=x;
   }
   printf("\n Final value of n is not sufficient \n");
   printf("To achieve desired accuracy\n");
   goto end;
   output:
      printf("\n Exit from loop \n");
      printf("SUM = %f; No. of terms = %d\n", sum,n);
      end:
         ;
}
…………………………………………………………………………………...
```

Output

```
Enter the value of x: .21

 Exit from loop
SUM = 1.265800; No. of terms = 7
Enter the value of x: .75

 Exit from loop
SUM = 3.999774; No. of terms = 34
```

Program 5.6.2 write a C program that demonstrate the inner and outer loop using goto statement.

```
#include <stdio.h>
void main()
{
   int p, q;
   for ( p = 0; p < 5; p++ )
   {
      printf( "Outer loop executing. p = %d\n", p );
      for ( q= 0; q < 2; q++ )
      {
         printf( " Inner loop executing. q = %d\n", q );
         if ( p == 5 )
            goto stop;
      }
   }
```

```
    /* This message does not print: */
    printf( "Loop exited. p = %d\n", p );

    stop:
        printf( "Jumped to stop. p = %d\n", p );
}
……………………………………………………………………………..
Output
Outer loop executing. p = 0
 Inner loop executing. q = 0
 Inner loop executing. q = 1
Outer loop executing. p = 1
 Inner loop executing. q = 0
 Inner loop executing. q = 1
Outer loop executing. p = 2
 Inner loop executing. q = 0
 Inner loop executing. q = 1
Outer loop executing. p = 3
 Inner loop executing. q = 0
 Inner loop executing. q = 1
Outer loop executing. p = 4
 Inner loop executing. q = 0
 Inner loop executing. q = 1
Loop exited. p = 5
Jumped to stop. p = 5
```

Activity

1. what will be the output of the following program segment:

```
void main () {
   int num = 10;
   LOOP: do {
      if( num = = 15) {
         num = num + 1;
         goto LOOP;
      }
      printf("value of Num: %d\n", num);
      num++;
   }while( num < 20 );
}
```

Summary Summary	

In this lesson

- We have learned about ***goto*** statement with its syntax.
- We have also learned how a ***goto*** statemet works in program.

ASSIGNMENT

Assignment

1. Write the importance of uses of goto statement in C program.
   ………………………………………………………………………………………
   ………………………………………………………………………………………
2. Find errors if any and correct them and mention the output of the following program segment after correcting the errors:

```
void main ()
{
    int num = 10;
    TEST  do{
                if( num = 14){
                    num = + 2;
                   goto;
                  }
            printf("value of a: %d\n", num);
            num++;
            }while( test < 20 )}
```

Assessment

Assessment

Write "T" for true and "F" for false the following sentences:

1. The ***goto*** statement is used to exit the normal sequence of C program execution.
2. The ***goto*** statement provides a conditional jump from the ***'goto'*** to a labeled statement in the same function.
3. The ***goto*** statement can transfer the program control to any place in a program.

Multiple Choice Questions (MCQ)

1. In goto statement the Label_name can be any ordinary text except C-

   a) Keyword b) Expression c) Variable d) Constant

2. The goto statement is used to change the normal sequence of C program execution by transferring-

   a) Control to any place of the program
   b) Control to only specified place of the program
   c) Control to end of the program
   d) None of these

3. In goto statement every labeled statement within the program must have a

   a) Different label b) unique label c) Multiple label d) None of these

4. The ***goto*** statement provides an unconditional jump from the ***'goto'*** to a labeled statement in the-

   a) Same function b) Same constant c) Same variable d) Multiple function

Exercises

1. Why is goto statement used in programming C?
2. Mention the benefits of using goto statement in C.
3. Explain goto statement with its syntax and discuss its flowchart.
4. Describe the difference among break, continue and goto statements with proper example.