MODULE 2

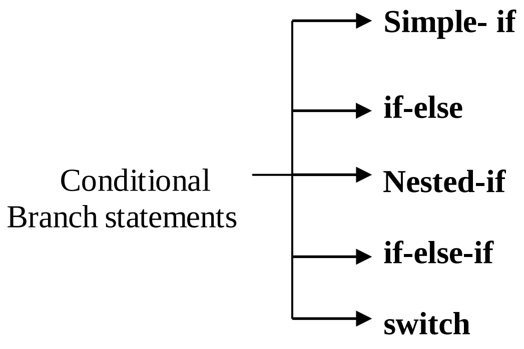
Chapter 10
Decision Control and Looping Statements

Introduction to Decision Control or Branching Statements

- ✓ **The statements that transfer the control from one place to other place in the program with or without any condition are called branch statements or selection statements.**
- ✓ The branching statements are classified into two types:
  - i. Conditional branch statements
  - ii. Unconditional branch statements

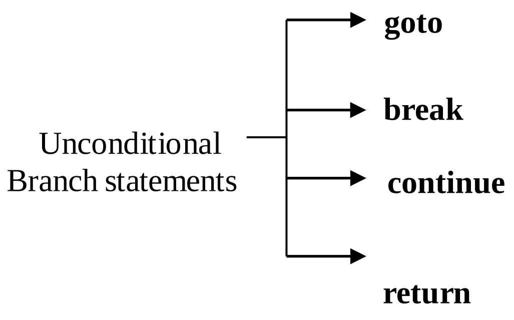
(i) Conditional branch statements

- ✓ **The statements that transfer the control from one place to another place in the program based on some conditions are called Conditional branch statements.**



(ii) Unconditional branch statements

- ✓ **The statements that transfer the control from one place to another place in the program without any condition are called Unconditional branch statements.**



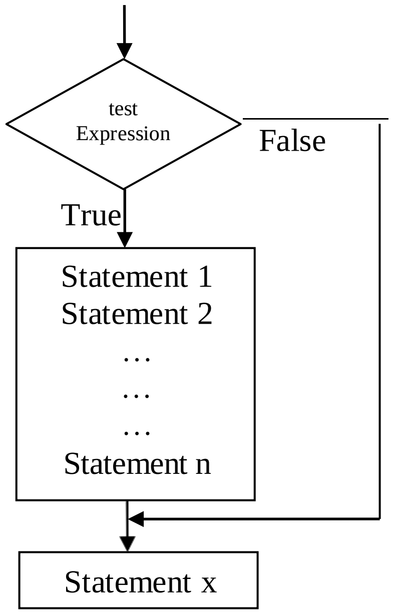
Conditional Branching Statements

if Statement

- ✓ The 'if' statement is the simplest form of decision control statement.
- ✓ When a set of statements have to be executed when an expression (condition) is evaluated to true or skipped when an expression (condition) is evaluated to false, then if statement is used.
- ✓ It is used whenever there is only one choice (alternative). Hence it is also called as **"One-way decision or selection statement".**

Syntax of if statement

```
if(test Expression)
{
   Statement 1;

   Statement 2;

         ...

         …

         …

   Statement n;

}

Statement x;
```



✓ The ‘if’ structure may include one statement or ‘n’ statements enclosed within curly brackets.

Working Principle

✓ First the test expression is evaluated. If the test expression is true, then the statements of ‘if’ block (statement 1 to n) are executed. Otherwise these statements will be skipped and the execution will jump to statement x.

Rules for if-statement

a) ‘if’ must be followed by an expression and the expression must be enclosed within parenthesis.
b) If multiple statements have to be executed when the expression is true, then all those statements must be enclosed within braces.
c) No semicolon is required for an if-statement.
 if(a<b); // statement indicates a NULL statement. It will ‘do nothing’.

Example Programs for ‘if’ statement

1. Write a C program to determine whether a person is eligible to vote or not.

```
#include<stdio.h>
void main()
{
      int age;
      printf(“Enter the age:”);
      scanf(“%d”,&age);
      if(age>=18)
          printf(“\nThe person is eligible to vote”);
      if(age<18)
          printf(“\nThe person is not eligible to vote”);
}
```

2. Write a C program to check whether the number is even or odd and print the appropriate message.

```
#include<stdio.h>
void main()
{
      int n;
      printf("Enter a number:");
      scanf("%d",&n);
      if(n%2==0)
          printf("Number is even");
      if(n%2=0)
          printf("Number is odd");
 }
```

3. Write a C program to print the largest of two numbers.

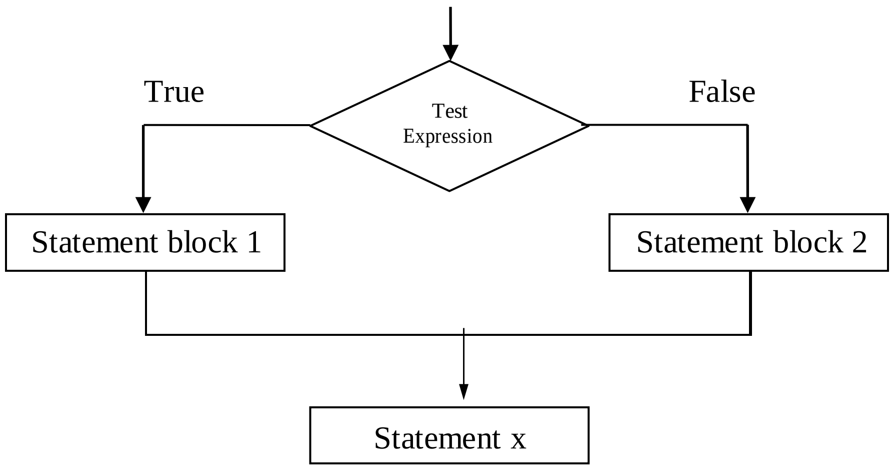
```
#include<stdio.h>
void main()
{
      int a,b;
      printf("Enter the two numbers:");
      scanf("%d%d",&a,&b);
      if(a>b)
          printf("\n a is greater than b");
      if(b>a)
          printf("\n b is greater than a");
 }
```

if-else statement

- ✓ If one set of activities have to be performed when an expression is evaluated to true and another set of activities have to be performed when an expression is evaluated to false, then if-else statement is used.
- ✓ The is-else statement is used when we must choose between two choices (alternatives). Hence is also called as **"Two-way Decision or Selection Statement".**

Syntax of if-else statement:

```
if(test Expression)
{
   Statement block 1;

 }

else

{

   Statement block 2;

 }

Statement x;
```



Working Principle

- ✓ According to the if-else construct, first the 'test expression' is evaluated.
- ✓ If the expression is true then Statement block 1 is executed and Statement block 2 is skipped.
- ✓ If the expression is false the Statement block 2 is executed and Statement block 1 is ignored.
- ✓ Now in any case after the Statement block 1 or 2 gets executed the control will pass to Statement x. It is executed in every case.

Example Programs for 'if-else' statement

1. Write a C program to determine whether a person is eligible to vote or not.

```
#include<stdio.h>
 void main()
{
      int age;
       printf("Enter the age:");
      scanf("%d",&age);
      if(age>=18)
          printf("\nThe person is eligible to vote");
      else
          printf("\nThe person is not eligible to vote");
   }
```

2. Write a C program to check whether the number is even or odd and print the appropriate message.

```
#include<stdio.h>
void main()
{
       int n;
       printf(“Enter a number:”);
       scanf(“%d”,&n);
       if(n%2==0)
           printf(“Number is even”);
       else
           printf(“Number is odd”);
 }
```

3. Write a C program to enter a character and then determine whether it is a vowel or not.

```
#include<stdio.h>
void main()
{
       char ch;
       printf(“Enter any character:”);
       scanf(“%c”,&ch);
       if(ch==’a’||ch==’e’||ch==’i’||ch==’o’||ch==’u’||ch==’A’||ch==’E’||ch==’I’||ch==’O’||ch==’U’)
           printf(“\n Character is a vowel”);
       else
           printf(“\n Character is a consonant”);
 }
```

4. Write a C program to find whether a given year is leap year or not.

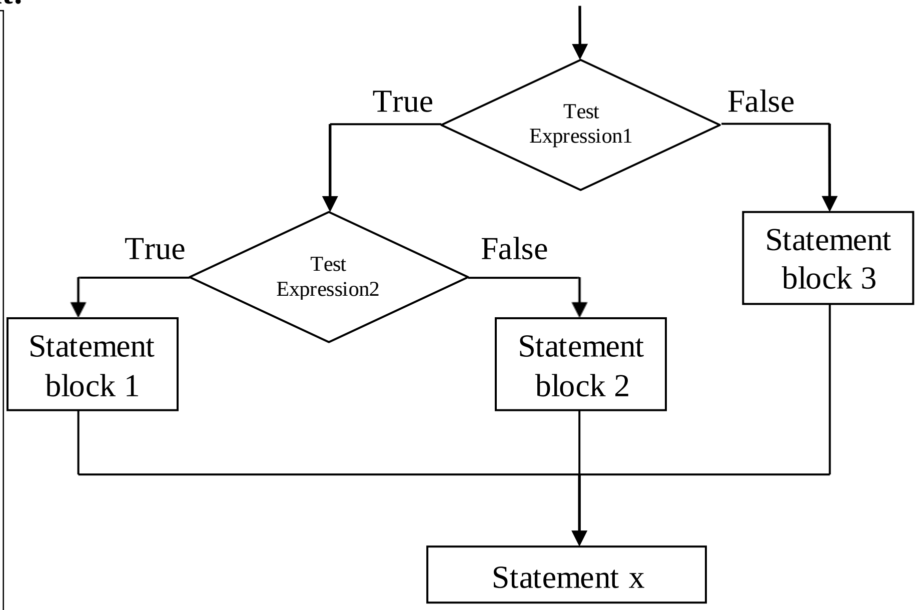
```
#include<stdio.h>
void main()
{
     int year;
     printf(“Enter any year:”);
     scanf(“%d”,&year);
     if(((year%4==0)&&(year%100!=0))||(year%400==0))
             printf(“%d is a leap year”,year);
     else
             printf(“%d is not a leap year”,year);
}
```

Nested if-else statement

- ✓ **An if-else statement within if statement or an if-else statement within else statement is called "nested if or if-else statement".**
- ✓ When an action has to be performed based on many decisions involving various types of expressions and variables, then this statement is used. So it is called as **"Multi-way decision statement".**

Syntax for nested if-else statement:

```
if(test Expression1)
{
   if(test Expression2)
  {
     Statement block 1;
   }
   else
   {
     Statement block 2;
    }
}
else

{

   Statement block 3;

 }

Statement x;
```



Working Principle

- ✓ If the test expression 1 is evaluated to true, then the test expression 2 is checked for true or false. If the test Expression2 is evaluated to true, then the statements in block 1 are executed, otherwise the statements in block 2 are executed. After executing the inner if-else the control comes out and the statement x is executed.
- ✓ If the test expression1 itself is false, then the statements in block 3 are executed. After executing these statements, the statement x is executed.

Example Programs for 'nested if-else' statement

1. Write a C program to find the largest of three numbers.

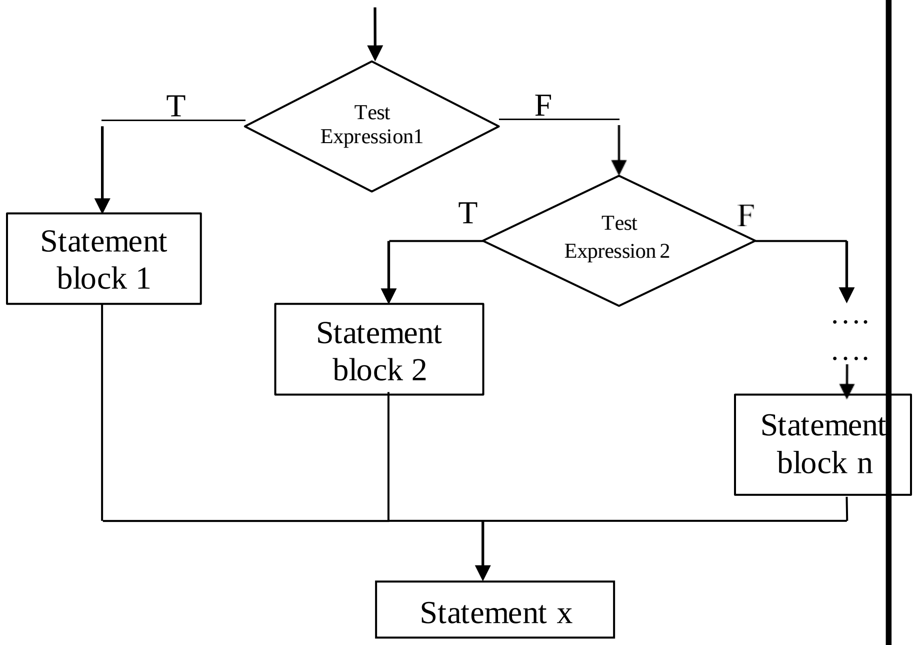
```
#include<stdio.h>
void main()
{
      int a,b,c;
      printf("Enter the three numbers:");
      scanf("%d%d%d",&a,&b,&c);
      if(a>b)
      {
         if(a>c)
            printf(“Max=%d”,a);
         else
            printf(“Max=%d”,c);
      }
      else
      {
         if(b>c)
            printf(“Max=%d”,b);
         else
            printf(“Max=%d”,c);
      }
}
```

if-else- if Statement

- ✓ It is a special case of nested-if statement where nesting takes place only in the else part.
- ✓ The orderly nesting of if-else statement only in the else part is called ‘else-if-ladder’.
- ✓ When an action has to be selected based on the range of values, then this statement is used. So it is called “Multi-way Decision or Selection Statement”.

Syntax :

```
if (test expression1)
{
   Statement block 1;
}
else if (test expression2)
{
   Statement block 2;
}
else
{

   Statement block n;

}

Statement x;
```



Working Principle

- ✓ The ‘Expressions’ are evaluated in order. If any expression is true then the statement associated with it is executed and this terminates the whole chain and statement x is executed.
- ✓ The last ‘else’ part is executed when all the test expression are false.

Example Programs for ‘cascaded if-else’ statement

1. Write a C program to demonstrate the use of cascaded if structure.

```
#include<stdio.h>
void main()
{
    int x,y;
    printf(“Enter the values of x and y:”);
    scanf(“%d%d”,&x,&y);
    if(x==y)
        printf(“\n The two numbers are equal”);
    else if(x>y)
        printf(“\n %d is greater than %d”,x,y);
    else
        printf(“\n %d is less than %d”,x,y);
 }
```

2. Write a C program to print whether the number is positive, negative or zero.

```
#include<stdio.h>
void main()
{
    int n;
    printf(“Enter a number:”);
    scanf(“%d”,&n);
    if(n==0)
        printf(“Number is zero”);
    else if(n>0)
        printf(“Number is positive”);
    else
        printf(“Number is negative”);
 }
```

3. Write a C program to input 3 numbers and then find the largest of them using ‘&&’ operator.

```
#include<stdio.h>
void main()
{
      int a,b,c;
      printf(“Enter the three numbers:”);
      scanf(“%d%d%d”,&a,&b,&c);
      if(a>b && a>c)
          printf(“\n %d is the largest number”,a);
      else if(b>a && b>c)
          printf(“\n %d is the largest number”,b);
      else
          printf(“\n %d is the largest number”,c);
 }
```

4. Write a C program to display the examination results.

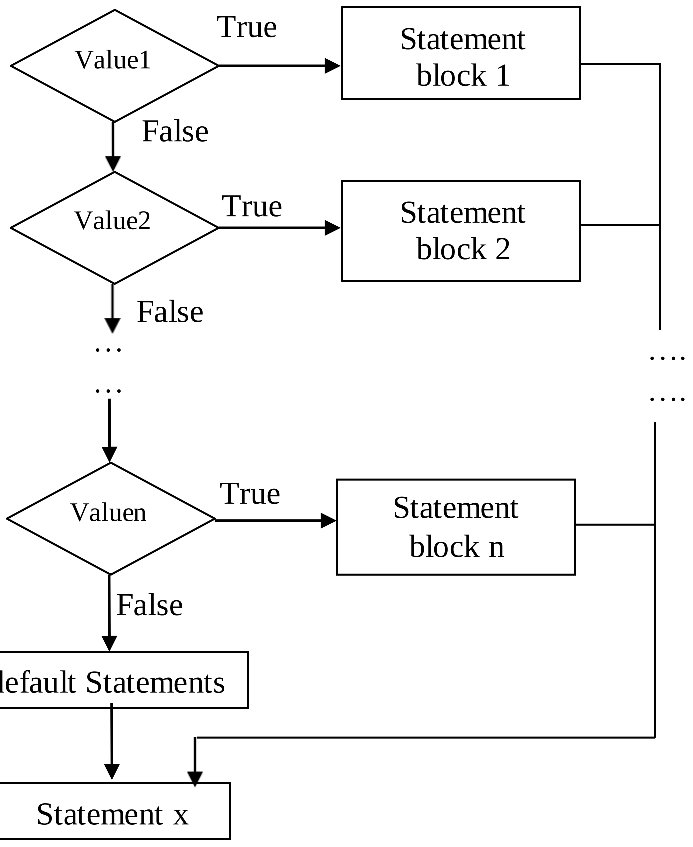
```
#include<stdio.h>
void main()
{
      int marks;
      printf(“Enter the marks:”);
      scanf(“%d”,&marks);
      if(marks>=75)
          printf(“\n First Class with Distinction”);
      else if(marks>=60&&marks<75)
          printf(“\n First Class”);
      else if(marks>=50&&marks<60)
          printf(“\n Second Class”);
      else if(marks>=40&&marks<50)
          printf(“\n Third Class”);
      else
          printf(“\n Fail”);
  }
```

switch statement

- ✓ The ‘switch’ statement is a control statement used to select one alternative among several alternatives.
- ✓ It is a **“multi-way decision statement”** that tests whether an expression matches one of the case values and branches accordingly.

switch statement syntax

```
switch(expression)
{
  case value-1: Statement block 1;
                break;
  case value-2: Statement block 2;
                break;
                  …..
                  …..

  case value-n: Statement block n;
                break;
  default:      Statements;

 }
Statement x;
```



Working Principle

- ✓ First the expression within switch is evaluated.
- ✓ The value of an expression is compared with all the case values.
- ✓ The value of an expression within switch is compared with the case value-1. If it matches then the statements associated with that case are executed. If not then the case value-2 is compared, if it matches then the associated statements are executed and so on.
- ✓ The "default" statements are executed when no match is found.
- ✓ A default is optional.
- ✓ The 'break' statement causes an exit from the switch. 'break' indicates end of a particular case and causes the control to come out of the switch. **[Significance of break within switch statement]**

Example Programs for 'switch' statement

1. Write a C program to display the grade.

```
#include<stdio.h>
void main()
{  char grade;
   printf("Enter the grade:");
   scanf("%c",&grade);
   switch(grade)
   {
          case ‘O’: printf(“Outstanding”);
                    break;
          case ‘A’: printf(“Excellent”);
                    break;
          case ‘B’: printf(“Good”);
                    break;
          case ‘C’: printf(“Fair”);
                    break;
          case ‘F’: printf(“Fail”);
                    break;
          default : printf(“Invalid grade”);
        }
       }
```

2. Write a C program to enter a number from 1 to 7 and display the corresponding day of the week using switch.

```
   #include<stdio.h>
   void main()
   {    int day;
      printf(“Enter any number:”);
      scanf(“%d”,&day);
      switch(day)
     {
        case 1: printf(“Sunday”);
                break;
        case 2: printf(“Monday”);
                break;
        case 3: printf(“Tuesday”);
                break;
        case 4: printf(“Wednseday”);
                break;
        case 5: printf(“Thursday”);
                break;
        case 6: printf(“Friday”);
                break;
        case 7: printf(“Saturday”);
                break;
        default : printf(“invalid input”);
      }
  }
```

Iterative Statements or Loops

- ✓ A set of statements may have to be repeatedly executed for a specified number of times or till a condition is satisfied.
- ✓ **The statements that help us to execute a set of statements repeatedly for a specified number of times or till a condition is satisfied are called as Iterative statements or looping constructs or loop control statements.**
- ✓ These statements are also called as Repetitive or Iterative Statements.

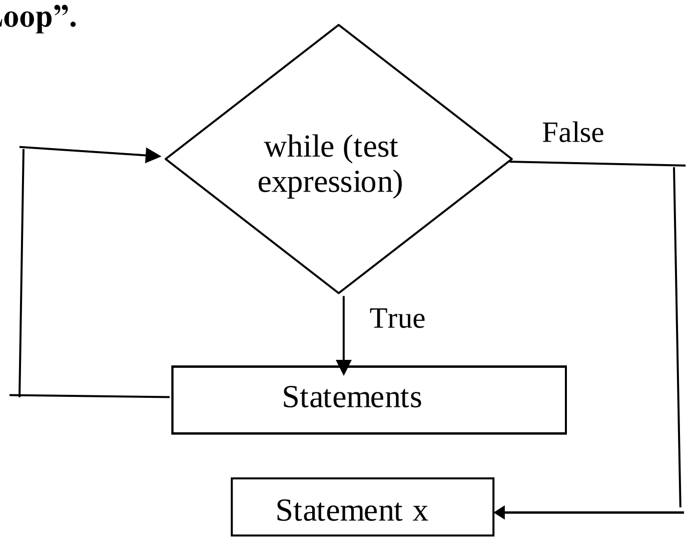
$$\begin{tikzcd}
 & \text{while loop} \\
\text{Loop Statements} \arrow[r] \arrow[ur] \arrow[dr] & \text{for loop} \\
 & \text{do-while loop}
\end{tikzcd}$$

while Loop

- ✓ **It is a control statement using which the programmer can give instructions to the computer to execute a set of statements repeatedly as long as specified condition is satisfied.**
- ✓ The expression is evaluated to TRUE or FALSE in the beginning of the while loop. Hence it is called as **"Entry-Controlled Loop".**

Syntax of while loop

```
while(test expression)
{
   Statements;
 }
Statement x;
```



Working principle

- ✓ The test expression is evaluated first, if it is TRUE then the set of statements within the body of the loop are executed repeatedly as long as specified test expression is TRUE.
- ✓ If the test expression is false, then the control comes out of the loop by skipping the execution of the statements within the body of the loop, by transferring the control to the Statement x.

Example programs of ‘while’ loop

1. Write a C program to print the natural numbers from 1 to 10.

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

2. Write a C program to print the natural numbers from 1 to n.

```
#include<stdio.h>
void main()
{
    int i=0,n;
    printf(“Enter the value of n:”);
    scanf(“%d”,&n);
    while(i<=n)
    {
            printf(“%d\n”,i);
            i++;
    }
}
```

3. Write a C program to print the natural numbers from n to 1.

```
#include<stdio.h>
void main()
{
    int n,i;
    printf(“Enter the value of n:”);
    scanf(“%d”,&n);
    i=n;
    while(i>=1)
    {
            printf(“%d\n”,i);
            i--;
    }
}
```

4. Write a C program to calculate the sum of the first 10 natural numbers.

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

```
{
      int i=1,sum=0;
      while(i<=10)
      {
              sum=sum+i;
              i++;
      }
       printf("\nSum=%d",sum);
  }
```

5. Write a C program to print the characters from A to Z.

```
#include<stdio.h>
void main()
{
        char ch= 'A';
        while(ch<= 'Z')
        {
                printf("%c\t",ch);
                ch++;
        }
}
```

6. Write a C program to find the factorial of a given number using while statement.

```
#include<stdio.h>
void main()
{
        int fact=1,i=1,n;
        printf("Enter the value of n:\n");
        scanf("%d",&n);
        while(i<=n)
        {
                fact=fact*i;
                i++;
         }
        printf("Factorial of a given number is=%d",fact);
 }
```

7. Write a C program to read a number and determine whether it is palindrome or not.

```
#include<stdio.h>
void main()
{       int digit,rev=0,num,n;
        printf("Enter the number:\n");
        scanf("%d",&num);
        n=num;
        while(num!=0)
        {
                digit=num%10;
                rev=rev*10+digit;
                num=num/10;
        }
        printf("The reversed number is=%d\n",rev);
        if(n==rev)
                printf(" The number is a palindrome");
        else
                printf("The number is not a palindrome");
}
```

8. Program to find GCD and LCM of 2 numbers.

```
#include<stdio.h>
void main()
{       int a,b,m,n,gcd,lcm,rem;
        printf("Enter the value for m and n :\n");
        scanf("%d %d", &a, &b);
        m=a;
        n=b;
        while(n!=0)
        {
                rem=m%n;
                m = n;
                n = rem;
        }
        gcd = m;
        lcm = ( a * b ) / gcd;
        printf("The GCD of the numbers is=%d\n", gcd );
        printf("The LCM of the  numbers is=%d\n", lcm );
}
```

OUT PUT:

```
Enter the value for m and n:
6
12
The GCD of 6 12 numbers is= 6
The LCM of 6 12 numbers is= 12
```

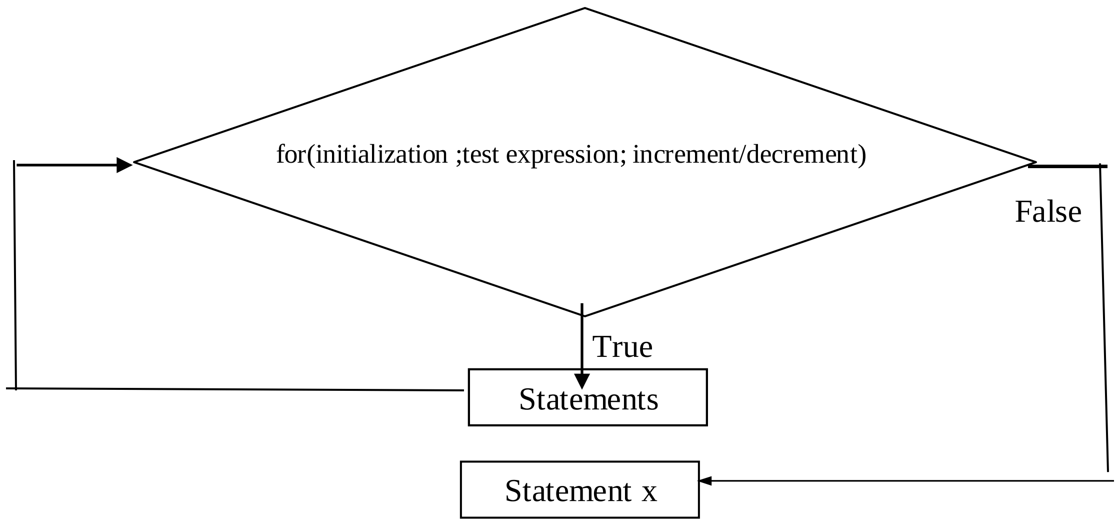
for Loop (Counter- Controlled Loop)

- ✓ **It is a control statement using which the programmer can give instructions to the computer to execute a set of statements repeatedly for a specific number of times.**
- ✓ It is also called **as entry-controlled loop** or **pretest**

loop.**Syntax of for loop**

```
for(initialization ;test expression; increment/decrement)
{
   Statements;

 }

Statement x;
```

Flowchart



Working Principle

- ✓ **initialization:** In this section loop variable is initialized, like i=0, n=0, i=1,n=1. (i and n are loop variables).
- ✓ **test expression:** The test expression may be a relational expression or logical expression or both, which is evaluated to TRUE or FALSE. Depending on the value of the test expression, the body of the loop is executed. If the test expression is TRUE, then the body of the loop is executed. This process of execution of body of the loop is continued as long as the expression is TRUE. When the test expression becomes FALSE, execution of the statements contained in the body of the loop are skipped, thereby transferring the control to the Statement x, which immediately follows the for loop.
- ✓ **increment/decrement:** This section increments or decrements the loop variables after executing the body of the loop.

Infinite Loop

- ✓ A for loop without a test expression is an Infinite loop.

```
Ex: for(i=0;    ;i++)
    {
            .........
    }                       is an "infinite" loop.
```

Example programs of ‘for’ loop

1. Write a C program to print the natural numbers from 1 to 10.

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

2. Write a C program to print the natural numbers from 1 to n.

```
#include<stdio.h>
void main()
{
    int i,n;
    printf("Enter the value of n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
             printf("%d\n",i);
    }
  }
```

3. Write a C program to compute the sum of the series 1+2+3+……+n.

```
#include<stdio.h>
void main()
{
        int i,sum=0,n;
        clrscr();
        printf("Enter the number of terms:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                sum=sum+i;
        }
         printf("\nSum of the series=%d",sum);
 }
```

4. Write a C program to generate ‘n’ Fibonacci numbers.

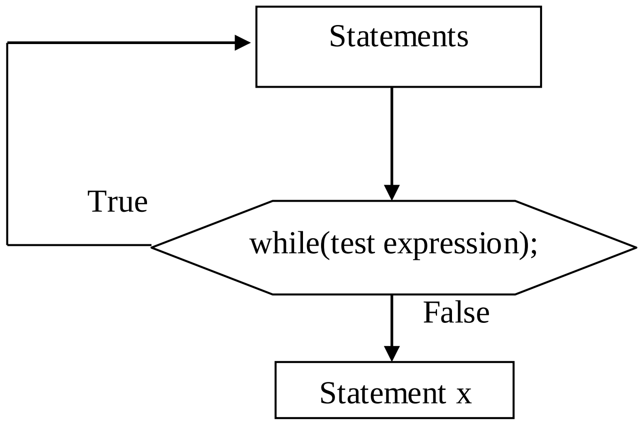
```
#include<stdio.h>
void main()
{
            int n,f1,f2,f3,i;
            f1=0;
            f2=1;
            printf(“Enter the value of n:”);
            scanf(“%d”,&n);
            if(n==1)
                      printf(“%d”,f1);
            else
           {
                      printf(“%d\t%d\t”,f1,f2);
                      for(i=3;i<=n;i++)
                      {

                        f3=f1+f2;
                        printf(“%d\t”,f3);f1=f2;
                        f2=f3;
                       }
                   }
                   }

            }
}
```

do-while Loop

- ✓ **do- while loop is used when a set of statements have to be repeatedly executed at least once.**
- ✓ Since the test expression is evaluated to TRUE or FALSE at the end of do-while loop, the do-while loop is called as **exit-controlled loop**.
- ✓ The while and for loop test the expression at the top.
- ✓ The do-while tests the expression at the bottom after making each passes through the loop body.

Syntax of do-while loop

```
do{
        Statements;

   }while(test expression);

 Statement x;
```



Working Principle

- ✓ It is a **post-test or bottom-testing loop** and hence the statements contained within the body of the loop are executed first and then the expression is evaluated to TRUE or FALSE. If it is TRUE, then the statements contained within the body of the loop are executed once again and the expression is evaluated. This is repeated until the expression is evaluated to FALSE.
- ✓ If the expression is FALSE, then the control comes out of the loop by skipping the execution of the statements within the body of the loop, by transferring the control to the Statement x.

2.6.1 Difference between while loop and do-while loop

	while loop	**do-while loop**
1	It is **entry controlled loop. (top-testing)**	It is **exit controlled loop**. **(bottom-testing)**
2	It is pre-test loop.	It is post-test loop.

3	**Syntax:**
	while(expression)
	{
	    Statements;
	}
		Syntax:
		do {
		    Statements;
		} while(expression);

4	**Working Principle:** If the expression is evaluated to true, then the statements within the body of loop are executed. If the expression is false, then the statements within the body of the loop are not executed.	**Working Principle:** On reaching do statement, it proceeds to execute the statements within the body of the loop irrespective of the test expression. If the expression is evaluated to true, then the statements within the body of loop are executed once again. If the expression is false, then the statements within the body of the loop are not executed.
5	In while loop, while statement does not end with semicolon (;).	In do-while loop, while statement ends with semicolon (;).

6	Ex:
	int i=0,sum=0;
	while(i<=n)
	{
	  sum=sum+i;
	  i=i+1;
	}
		Ex:
		int i=0,sum=0;
		do{
		        sum=sum+i;
		        i=i+1;
		}while(i<=n);

Example programs of ‘do-while’ loop

1. Write a C program to print the natural numbers from 1 to n.

```
#include<stdio.h>
void main()
{
    int i=1,n;
    printf(“Enter the value of n:”);
    scanf(“%d”,&n);
    do
    {
            printf(“%d\n”,i);
            i++;
    }while(i<=n);
}
```

2. Write a C program to calculate the sum and average of first ‘n’ numbers.

```
    #include<stdio.h>
    void main()
    {
             int n,i=0,sum=0;
             float avg=0.0;
             printf(“Enter the value of n:”);
             scanf(“%d”,&n);
            do
            {
                    sum=sum+i;
```

```
                i++;

            }while(i<=n);
            avg=sum/n;
            printf("The sum of numbers=%d",sum);
            printf("The average of numbers=%f",avg);
    }
```

Nested Loops

- ✓ The loops that can be placed inside other loops.
- ✓ It will work with any loops such as while, do-while and for.
- ✓ Nested loop is commonly used with the for loop because this is easiest to control.
- ✓ The 'inner for loop' can be used to control the number of times that a particular set of statements will be executed.
- ✓ The 'outer for loop' can be used to control the number of times that a whole loop is repeated.

Example Program: Write a C program to print the following output.

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

```
        #include<stdio.h>
        void main( )
        {
            int i,j;
        --> for(i=1;i<=5;i++)
            {
                printf("\n");
Outer____       for(j=1;j<=i;j++)       <--
for loop        {                          |-- Inner
                    printf("%d",j);        |   for loop
                }                       <--
        -->  }
        }
```

Unconditional Branch statements

break statement

- ✓ **In C, break statement is used to terminate the execution of the nearest enclosing loop in which it appears.**
- ✓ It is 'jump' statement which can be used in **switch statement and loops**.
- ✓ **The break statement in switch statement causes the control to come out of the switch statement and transfers the control to the statement which follows the switch statement.**

Ex:

```
switch(ch)
{
 case 1: printf("1st statement");
         break;
 case 2: printf("2nd  statement");
         break;
 default: printf("nth statement");
}
```

If 'case 1' is selected by the programmer then the output will be 1st statement only. It will directly come out of the switch.

- ✓ **If break is executed in a loop (for/while/do-while) then the control comes out of the loop and the statement following the loop will be executed.**

Syntax

1. while(…)
```
{
 ……
 if(condition)
    break;
……
}
……
```
Transfers the control out of the while loop

2. do
```
{
 ……
 if(condition)
    break;
……
}while(condition);
……
```
Transfers the control out of the do-while loop

```
3.  for(…)
    {
      ……
      if(condition)
          break;
     ……
    }                    Transfers the control out of the for loop
     ……
```

Example programs of ‘break’ statement

1. Write a C program to print the numbers (0 to 4) using break.

```
#include<stdio.h>
void main()
{
         int i=0;
        while(i<=10)
        {
                if(i==5)
                     break;
                printf(“%d\n”,i);
                i=i+1;
        }
}
```

continue statement

- ✓ During execution of a loop, it may be necessary to skip a part of the loop based on some condition. In such cases, we use continue statement.
- ✓ **The continue statement is used only in the loops to terminate the current iteration and continue with the remaining iterations.**

Syntax:

```
1.   while(… )
    {
      if(condition)          Transfers the control to the expression of the while loop
        continue;
      ……….
    }
     ……….
```

```
2.  do
    {
        ……
        if(condition)
           continue;
       ……                    Transfers the control to the expression of the do-while loop
     }while(condition);
```

```
      ……
3.  for(…)
    {
        ……
        if(condition)
            continue;
      ……
    }
      ……
```

Transfers the control to the expression of the for loop

Example programs of ‘continue’ statement

1. Write a C program to display the output in the following form 1 3 4 5

```
#include<stdio.h>
void main()
{
        int i;
        for(i=1;i<=5;i++)
        {
                if(i==2)
                    continue;
                printf("%d",i);
         }

}
```

Output: 1 3 4 5 [if i==2 then the continue statement is executed and the statements following continue are skipped]

goto statement

- ✓ **The goto statement is a ‘jump’ statement that transfers the control to the specified statement (Label) in a program unconditionally.**
- ✓ The specified statement is identified by ‘label’(symbolic name). Label can be any valid variable name that is followed by a colon (:).

Syntax

```
goto label;
    …………
    ………….
label:
    Statements
```

Forward Jump

```
label:
  Statements
   …………
   …………
goto label;
```

Backward Jump

Example programs of ‘goto’ statement

1. Write a C program to calculate the sum of all numbers entered by the user.

```
#include<stdio.h>
void main()
{
        int n,i,sum=0;
        printf(“Enter the number:\n”);
        scanf(“%d”,&n);
        sum=i=0;
        top:    sum=sum+n;
                i=i+1;
       if(i<=n)         goto top;
                printf(“Sum of Series=%d”,sum);

 }
```