# MODULE 2: Branching and Looping

I. Statements in C are of following types:

1. **Simple statements**: Statements that ends with semicolon
2. **Compound statements**: are also called as block. Statements written in a pair of curly braces.
3. **Control Statements**: The statements that help us to control the flow of execution in a program.
   There are two types of control statements in C
   a. **branching** b. **looping**
   Branching is deciding what actions to take and looping is deciding how many times to take a certain action.
   a. **Branching statements:** The statements that help us to jump from one statement to another statement with or without condition is called **branching statements**. These statements are also called as **selection statements** since they select some statements for execution and skip other statements. These statements are also called as **decision making statements** because they make decision based on some condition and select some statements for execution.
      Branching is so called because the program chooses to follow one branch or another.
      Branching Statements are of two types:
      i) **Conditional Branching statements**: The statements that help us to jump from one statement to another statement based on condition.
         **Example**:
         a. simple if statement,
         b. if…else statement,
         c. nested if…else statement,
         d. else…if ladder (cascaded if statement), and
         e. switch statement
      ii) **Unconditional Branching statements**: The statements that help us to jump from one statement to another statement without any conditions.
         **Example**:
         a. goto statement,
         b. break statement,
         c. continue statement and
         d. return statement
   b. **Looping statements:** The statements that help us to execute set of statements repeatedly are called as **looping statements**.
      **Example:**
      a. *while* statement,

b. *do…while* statement and
c. *for* statement

**i. Conditional Branching Statements**

The statements that help us to jump from one statement to another statement based on condition.

## a. Simple *if* Statement

This is basically a "one-way" decision statement.
This is used when we have only one alternative.
It takes an expression in parenthesis and a statement or block of statements. If the expression is **TRUE** then the statement or block of statements gets executed otherwise these statements are skipped.
NOTE: Expression will be assumed to be **TRUE** if its evaluated value is **non-zero**.
The **syntax** is shown below:

```
if (expression)
{
        statement1;
}
Next-Statement;
```

Where,

Expression can be arithmetic expression, relational expression, logical expression, mixed mode expression, constant or an identifier. Firstly, the expression is evaluated to true or false.
If the **expression** is evaluated to **TRUE**, then **statement1** is executed and **jumps** to Next-Statement for execution.
If the **expression** is evaluated to **FALSE**, then **statement1** is skipped and **jumps** to Next-Statement for execution.

The flow diagram (Flowchart) of simple if statement is shown below:

**Example:** Program to illustrate the use of if statement.
**/* Program to find a given number is positive */**

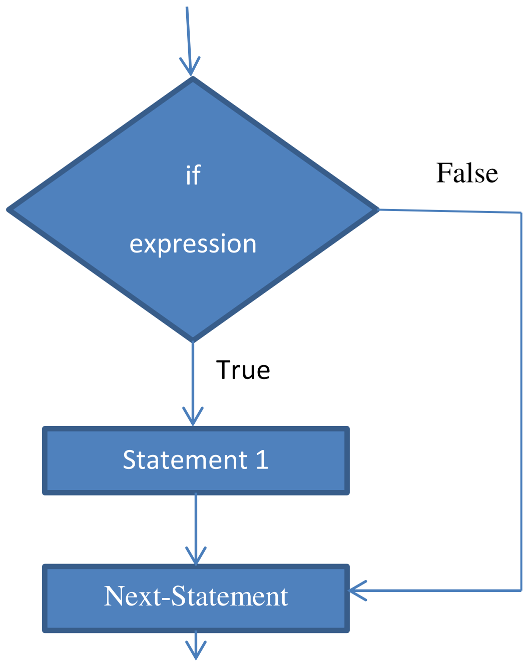
```
#include<stdio.h>
int main()
{
        int n;
        printf("Enter any non-zero integer: \n");
        scanf("%d", &n);
        if(n>0)
                printf("Number is positive number ");
        return 0;
}
```

**Output:**

```
Enter any non-zero integer:
7
Number is positive number
```

In the above example, the program reads a integer number from user and checks is it greater than zero if true the it displays a message "Number is positive number" otherwise it does nothing. Hence, it considers only one alternative.

**Flowchart of if statement**



## b. THE *if…else* STATEMENT

This is basically a "two-way" decision statement. This is used when we must choose between two alternatives.

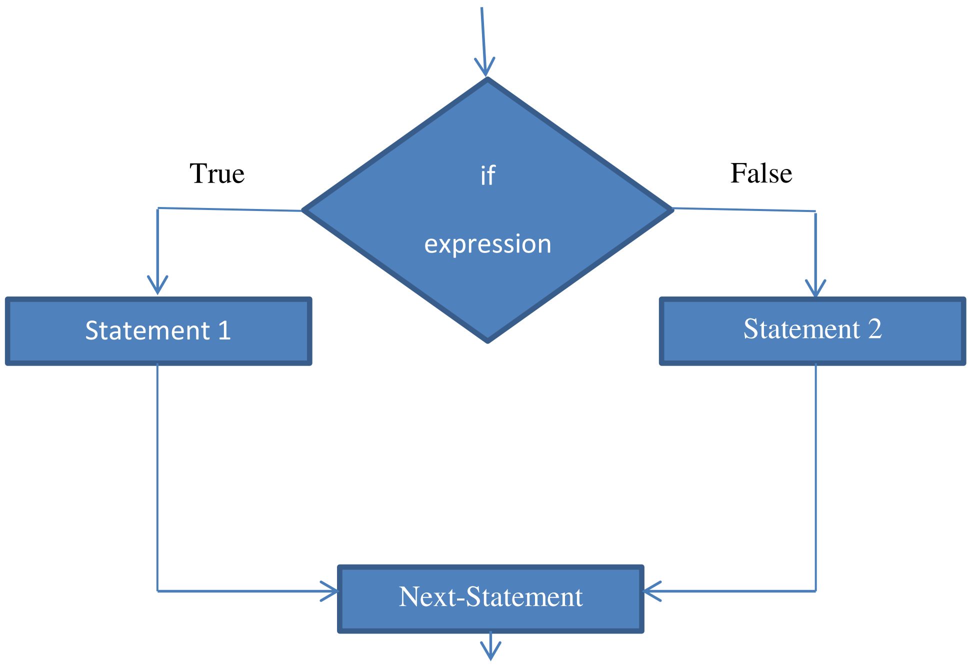
The syntax is shown below:

```
if(expression)
{
        statement1;
}
else
{
        statement2;
}
Next-Statement;
```

Where,

Expression can be arithmetic expression, relational expression, logical expression, mixed mode expression, constant or an identifier. Firstly, the expression is evaluated to true or false.

If the **expression** is evaluated to **TRUE**, then **statement1** is executed and **jumps** to Next-Statement (Skip statement2) for execution.
If the **expression** is evaluated to **FALSE**, then **statement2** is executed (**Skip statement1**) and **jumps** to Next-Statement for execution.
The flow diagram is shown below:



Example: Program to illustrate the use of if else statement.

```
#include<stdio.h>
int main()
{
        int n;
        printf("Enter any non-zero integer: \n") ;
        scanf("%d", &n)
        if(n>0)
                printf("Number is positive number");
        else
                printf("Number is negative number");
        return 0;
}
```

**Output 1:**

```
Enter any non-zero integer:
7
Number is positive number
```

**Output 2:**

```
Enter any non-zero integer:
-7
Number is negative number
```

In the above example, the program reads an integer number and prints "Number is positive number" is a given number "n" is greater than 0 otherwise it prints “Number is negative number”. Hence, it selects only one from two alternatives, so only it is called as two-way selection.

## c. THE nested if STATEMENT

An if-else statement is written within another if-else statement is called nested if statement. This is used when an action has to be performed based on many decisions. Hence, it is called as **multi-way decision statement**.

The syntax is shown below:

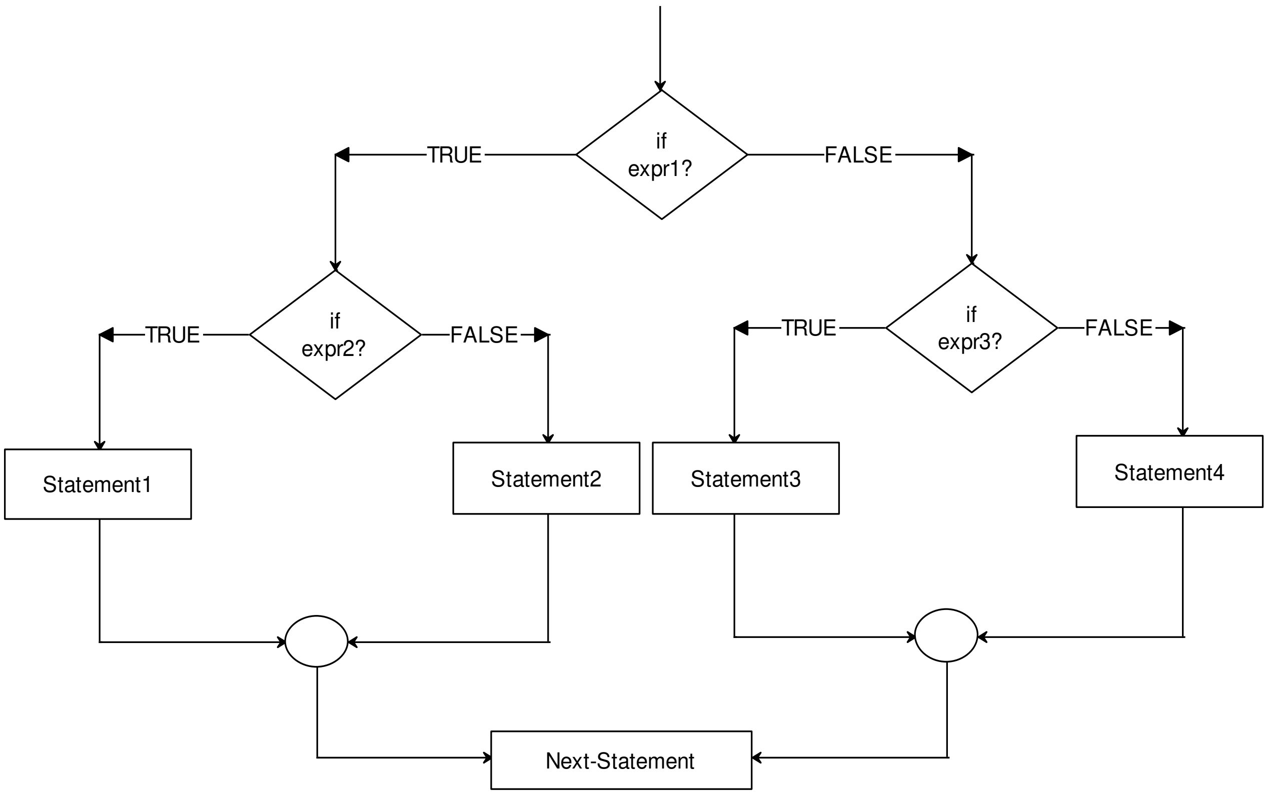
```
if(expr1)
{
        if(expr2)
                statement1;
        else
                statement2;
}
else
{
        if(expr3)
                statement3
        else
                statement4
}
Next-Statement;
```

• Here, firstly expr1 is evaluated to true or false.

```
        If the expr1 is evaluated to TRUE,
        then
                expr2 is evaluated to true or false.
                If the expr2 is evaluated to TRUE,
                then
                        statement1 is executed.
                If the expr2 is evaluated to FALSE,
                then
                        statement2 is executed.
        If the expr1 is evaluated to FALSE,
        then
                expr3 is evaluated to true or false.
                If the expr3 is evaluated to TRUE,
                then
                        statement3 is executed.
                If the expr3 is evaluated to FALSE,
                then
                        statement4 is executed.
```

The one of the drawback of this nested if statement is as the number of levels of nesting increases, the complexity increases and it makes difficult to understand and trace the flow of execution.

• The flow diagram is shown below:



**Example: Program to select and print the largest of the 3 numbers using nested "if-else" statements.**

```
#include<stdio.h>
int main()
{
        int a,b,c;
        printf("Enter Three Values: \n");
        scanf("%d %d %d ", &a, &b, &c);
        printf("Largest Value is: ") ;
        if(a>b)
        {
                if(a>c)
                        printf(" %d ", a);
                else
                        printf(" %d ", c);
        }
        else
        {
            if(b>c)
                printf(" %d", b);
            else
                printf(" %d", c);
        }
        return 0;
}
```

**Output:**

```
Enter Three Values:
17 18 6
Largest Value is: 18
```

**d. THE CASCADED if-else STATEMENT (THE else if LADDER STATEMENT)**

This is basically a "multi-way" decision statement. This is used when we must choose among many alternatives. This statement is alternative for nested if statement to overcome the complexity problem involved in nested if statement.

This statement is easier to understand and trace the flow of execution.

The **syntax** is shown below:

```
if(expression1)
{
        statement1;
}
else if(expression2)
{
        statement2;
}
else if(expression3)
{
        statement3
}
.
.
.
else if(expressionN)
{
        statementN;
else
{
        default statement;
}
Next-Statement;
```

The expressions are evaluated in order (i.e. top to bottom).

If an **expression** is evaluated to **TRUE**, then

→ Statement associated with the expression is executed &

→ Control comes out of the entire else if ladder

For example,

if **exprression1** is evaluated to **TRUE**,

then **statement1** is executed and jumps out of entire else...if ladder.

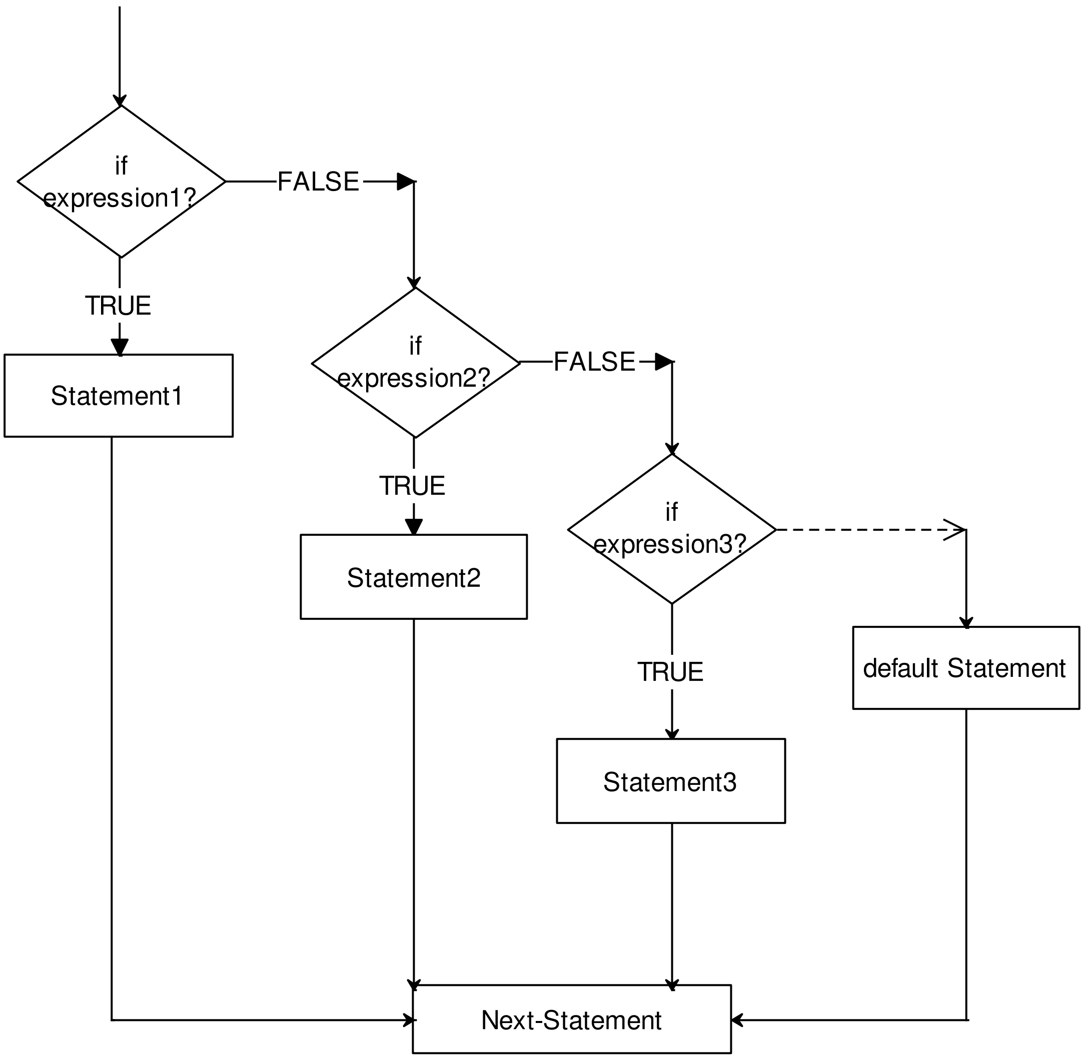
llly, if **exprression2** is evaluated to **TRUE**,

then **statement2** is executed and jumps out of entire else...if ladder.

If **all the expressions** are evaluated to **FALSE**,

then last statement (**default statement**) is executed.

**Flowchart is shown below:**



**Example: Program to select and print the largest of the 3 numbers using "else…if ladder" statements.**

```
#include<stdio.h>
int main()
{
        int a,b,c;
        printf("Enter Three Values: \n");
        scanf("%d %d %d ", &a, &b, &c);
        printf("Largest Value is: ") ;
        if ( a>b && a>c )
                printf(" %d ", a);
        else if ( b>a && b>c )
                printf(" %d ", b);
        else
                printf(" %d", c);
        return 0;
}
```

**Output:**

```
Enter Three Values:
17 18 6
Largest Value is: 18
```

e. **switch Statement**:

This is a multi-branch statement similar to the else…if ladder (with limitations) but clearer and easier to code. This is used when we must choose among many alternatives. This statement is to be used only when we have to make decision using integral values (i.e., either integer values or characters).

**Why we should use Switch statement?**

1. One of the classic problems encountered in nested if-else / else-if ladder is called problem of Confusion.
2. It occurs when no matching else is available for if.
3. As the number of alternatives increases the Complexity of program increases drastically.

To overcome these, C Provides a multi-way decision statement called 'Switch Statement'

**Syntax**:

```
switch ( expression )
{
        case label1 :
                        statement1 ;
                        break ;
        case label2 :
                        statement2 ;
                        break ;
        ...
        default :   statement ;
}
```

Where,

**Expression** can be either arithmetic expression that reduces to integer value, character constant, integer constant or an identifier of type integer.

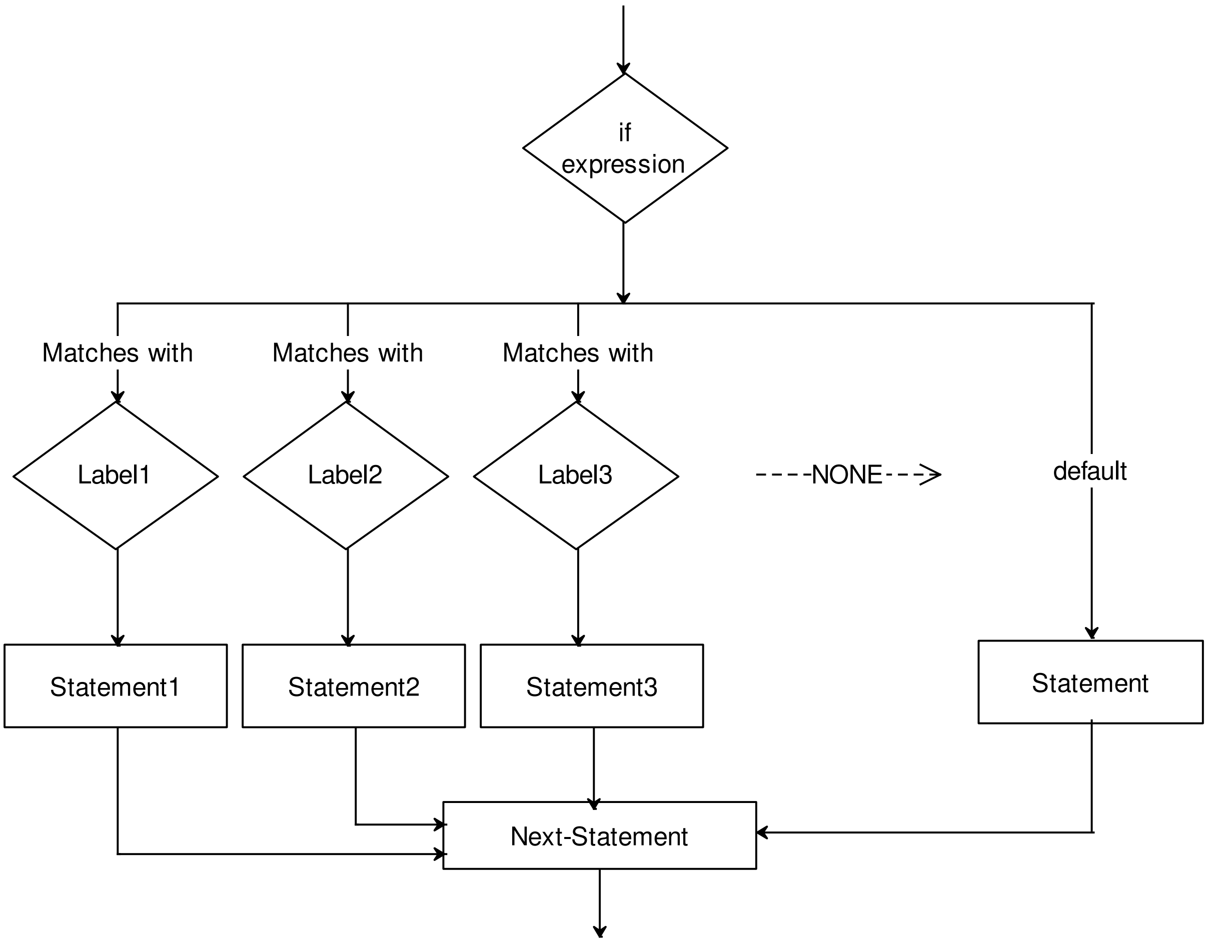
**Labels** can be either arithmetic expression that reduces to integer value, character constant or integer constant. But it cannot be an identifier.

The value of expression is tested for equality against the values of each of the labels specified in the case statements in the order written until a match is found. The statements associated with that case statement are then executed until a break statement or the end of the switch statement is encountered.

When a break statement is encountered execution jumps to the statement immediately following the switch statement.

The default section is optional -- if it is not included the default is that nothing happens and execution simply falls through the end of the switch statement.

The flow of switch statement is shown below:



The points to remember / Rules while using switch statement

- Can only test for equality with integer constants in case statements.
- All labels must be unique
- Character constants are automatically converted to integer.
- Floating point values are not allowed in expression and labels
- Break statement takes control out of the entire switch statement.
- Break Statement is Optional.

- Case Labels must end with Colon
- Case labels must have constants / constant expression
- Case label must be of integral Type ( Integer, or Character)
- Switch case should have at most one default label
- default label is Optional
- default label can be placed anywhere in the switch
- Two or more cases may share one break statement
- Nesting (switch within switch) is allowed.
- Relational Operators and logical operators are not allowed in Switch Statement.
- Empty Switch case is allowed.
- Macro Identifiers are allowed as Switch Case Label.

  **Example:**

  ```
  #define MAX 2          /* defines a MAX with constant value 2, as pre-
  processor #define will replace occurrence of MAX by constant value i.e 2
  therefor it is allowed. */
  switch(num)
  {
          case MAX:
                          printf("Number = 2");
                          break;
  }
  ```
- Const Variable is allowed in switch Case Statement.

  ```
  const int var = 2;       /* it defines var as constant variable and whose value
  cannot be changed, hence this variable can be used as case label */
  switch(num)
  {
          case var:                        // var is constant hence it is allowed
                          printf("Number = 2");
                          break;
  }
  ```

**For Example** :- **Program to simulate a basic calculator.**

```
#include <stdio.h>
int main()
{
        int num1, num2, result ;
        char op ;
        printf ( " Enter number operator number\n" ) ;
        scanf ("%f %c %f", &num1, &op, &num2 ) ;
        switch ( op )
        {
                case '+' :  result = num1 + num2 ;
```

```
                    break ;
        case '-' :  result = num1 - num2 ;
                    break ;
        case '*':  result = num1 * num2 ;
                    break ;
        case '%': result=num1%num2;
                    break;
        case '/' :  if ( num2 != 0.0 )
                    {
                          result = num1 / num2 ;
                          break ;
                    }
                    // else we allow to fall through for error message
        default :  printf ("ERROR -- Invalid operation or division by 0.0" ) ;
    }
    printf( "%f %c %f = %f\n", num1, op, num2, result) ;
    return 0;
}
```

Note : The break statement need not be included at the end of the case statement body if it is logically correct for execution to fall through to the next case statement (as in the case of division by 0.0) or to the end of the switch statement (as in the case of default : ).

**Output:**

```
Enter number operator number
2 + 4
2 + 4 = 6
```

II. **Ternary operator** (?:)

This operator operates on three operands hence the name ternary. This is a special shorthand operator in C and replaces if…else statement

Syntax of ternary operator is:

```
Expression1 ? expression2 : expression3;
```

If **expression1** is evaluated **TRUE**, then returns the result of expression2. i.e., it selects expression2 for execution and skips expression3.

If **expression1** is evaluated **FALSE**, then returns the result of expression3. i.e., it selects expression3 for execution and skips expression2.

Hence, it is same as TWO-WAY selection, it selects among two alternatives. Therefore any if…else statement can be replaced with this ternary operator.

For example,

```
if ( expression1 )
        expression2;
else
        expression3;
```

can be replaced with the more elegant form using ternary operator:

```
expression1 ? expression2 : expression3 ;
```

The ?: operator is a ternary operator in that it requires three arguments. One of the advantages of the ?: operator is that it reduces simple conditions to one simple line of code which can be thrown unobtrusively into a larger section of code.

**For Example :- to get the maximum of two integers, x and y, storing the larger in max.**

```
if ( x > = y )
        max = x ;
else
        max = y ;
```

The alternative to this could be as follows

```
max = x >= y ? x : y ;
```

giving the same result and it is a little bit more short.

## Programming examples on branching statements and ternary operator:

1. **Write a C program (WACP) to find largest of two numbers using if…else statement.**

   **Program:**

```
#include<stdio.h>
int main()
{
        int A, B, large;
        printf("Enter two numbers for A and B\n");
        scanf("%d%d" , &A, &B);
        if(A>B)
                large=A;
        else
                large=B;
        printf("Largest of %d and %d = %d\n",A,B,large);
        return 0;
}
```

2. **Write a C program (WACP) to find smallest of two numbers using if…else statement.**

   **Program:**

```
#include<stdio.h>
int main()
{
        int A, B, small;
        printf("Enter two numbers for A and B\n");
        scanf("%d%d" , &A, &B);
        if(A<B)
```

```
                small=A;
        else
                small=B;
        printf(“Smallest of %d and %d = %d\n” , A, B, small);
        return 0;
}
```

3. **WACP to find a given year is leap year or not using if…else statement.**

   **Program:**

```
#include<stdio.h>
int main()
{
        int year;
        printf(“Enter a year\n”);
        scanf(“%d” , &year);
        if(year%4 = = 0 && year%100 != 0 || year%400 = = 0)
                printf(“%d is a Leap Year\n” , year);
        else
                printf(“%d is Not a Leap Year\n” , year);
        return 0;
}
```

4. **WACP to determine a given number is even or odd using if…else statement.**

   **Program:**

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

5. **WACP to determine a given number is positive or negative using if…else statement.**

**Program:**

```
#include<stdio.h>
int main()
{
        int number;
        printf("Enter a number\n");
        scanf("%d" , &number);
        if(number >= 0)
                printf("%d is a Positive Number\n" , number);
        else
                printf("%d is Negative Number\n" , number);
        return 0;
}
```

6. **Write a C program (WACP) to find largest of two numbers using ternary operator.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, large;
        printf("Enter two numbers for A and B\n");
        scanf("%d%d" , &A, &B);
        large=(A>B) ? A : B ;
        printf("Largest of %d and %d = %d\n",A,B,large);
        return 0;
}
```

7. **Write a C program (WACP) to find smallest of two numbers using ternary operator.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, small;
        printf("Enter two numbers for A and B\n");
        scanf("%d%d" , &A, &B);
        small=(A<B) ? A : B ;
        printf("Smallest of %d and %d = %d\n" , A, B, small);
        return 0;
}
```

**8. WACP to find a given year is leap year or not using ternary operator.**

**Program:**

```
#include<stdio.h>
int main()
{
        int year;
        printf("Enter a year\n");
        scanf("%d" , &year);
        (year%4 = = 0 && year%100 != 0 || year%400 = = 0) ? printf("%d is a Leap
Year\n" , year) : printf("%d is Not a Leap Year\n" , year);
        return 0;
}
```

**9. WACP to determine a given number is even or odd using ternary operator.**

**Program:**

```
#include<stdio.h>
int main()
{
        int number;
        printf("Enter a number\n");
        scanf("%d" , &number);
        (number%2 = = 0) ? printf("%d is a EVEN Number\n" , number) : printf("%d
is ODD Number\n" , number);
        return 0;
}
```

**10. WACP to determine a given number is positive or negative using ternary operator.**

**Program:**

```
#include<stdio.h>
int main()
{
        int number;
        printf("Enter a number\n");
        scanf("%d" , &number);
        (number >= 0) ? printf("%d is a Positive Number\n" , number) : printf("%d is
Negative Number\n" , number);
        return 0;
}
```

**11. Write a C program (WACP) to find largest of three numbers using nested if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, C, large;
        printf("Enter three numbers for A , B and C\n");
        scanf("%d %d %d" , &A, &B, &C);
        if(A>B)
        {
                if(A>C)
                        large=A;
                else
                        large=C;
        }
        else
        {
                if(B>C)
                        large=B;
                else
                        large=C;
        }
        printf("Largest of %d, %d and %d is = %d\n" , A, B, C, large);
        return 0;
}
```

**12. Write a C program (WACP) to find smallest of three numbers using nested if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, C, small;
        printf("Enter three numbers for A , B and C\n");
        scanf("%d %d %d" , &A, &B, &C);
        if(A<B)
        {
                if(A<C)
                        small=A;
                else
                        small=C;
        }
        else
        {
                if(B<C)
                        small=B;
                else
                        small=C;
        }
        printf("Smallest of %d, %d and %d is = %d\n" , A, B, C, small);
        return 0;
}
```

**13. Write a C program (WACP) to find largest of three numbers using else…if ladder or cascaded if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, C, large;
        printf("Enter three numbers for A , B and C\n");
        scanf("%d %d %d" , &A, &B, &C);
        if( A>B && A>C )
                large = A;
        else if( B>A && B>C )
                large = B;
        else
                large = C;
        printf("Largest of %d, %d and %d is = %d\n" , A, B, C, large);
        return 0;
}
```

**14. Write a C program (WACP) to find smallest of three numbers using else…if ladder or cascaded if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, C, small;
        printf("Enter three numbers for A , B and C\n");
        scanf("%d %d %d" , &A, &B, &C);
        if( A<B && A<C )
                small = A;
        else if( B<A && B<C )
                small = B;
        else
                small = C;
        printf("Smallest of %d, %d and %d is = %d\n" , A, B, C, small);
        return 0;
}
```

**15. WACP to determine a given number is positive, negative or zero using nested if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int number;
        printf("Enter a Number\n");
        scanf("%d" , &number);
        if ( number >= 0 )
        {
                if( number > 0)
                        printf("%d is a Postive number\n", number);
                else
                        printf(" %d is ZERO\n", number);
        }
        else
                printf("%d is a Negative Number\n", number);
        return 0;
}
```

**16. WACP to determine a given number is positive, negative or zero using else…if ladder or cascaded if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int number;
        printf("Enter a Number\n");
        scanf("%d" , &number);
        if ( number > 0)
                printf("%d is a Postive number\n", number);
        else if ( number < 0)
                printf("%d is a Negative Number\n", number);
        else
                printf(" %d is ZERO\n", number);
        return 0;
}
```

**17. Write a C program (WACP) to find largest of three numbers using ternary operator.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, C, large;
        printf("Enter three numbers for A , B and C\n");
        scanf("%d %d %d" , &A, &B, &C);
        large = (A>B) ? ( A>C ? A : C ) : ( B>C ? B : C ) ;
        printf("Largest of %d, %d and %d is = %d\n" , A, B, C, large);
        return 0;
}
```

**18. Write a C program (WACP) to find smallest of three numbers using ternary operator.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B, C, small;
        printf("Enter three numbers for A , B and C\n");
        scanf("%d %d %d" , &A, &B, &C);
        small = (A<B) ? ( A<C ? A : C ) : ( B<C ? B : C ) ;
        printf("Smallest of %d, %d and %d is = %d\n" , A, B, C, small);
        return 0;
}
```

**19. WACP to read two numbers A and B and to print whether A is larger than, smaller than or equal to B using nested if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B;
        printf("Enter two numbers for A  and B \n");
        scanf("%d %d" , &A, &B);
        if (A >= B)
        {
                if ( A > B )
                        printf("%d is Larger than %d\n", A, B);
                else
                        printf("%d is Equal to %d\n", A, B);
        }
        else
                printf("%d is Smaller than %d\n", A, B);
        return 0;
}
```

**20. WACP to read two numbers A and B and to print whether A is larger than, smaller than or equal to B using else…if ladder or cascaded if statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B;
        printf("Enter two numbers for A  and B \n");
        scanf("%d %d" , &A, &B);
        if ( A > B )
                printf("%d is Larger than %d\n", A, B);
        else if (A < B)
                printf("%d is Smaller than %d\n", A, B);
        else
                printf("%d is Equal to %d\n", A, B);
        return 0;
}
```

**21. WACP to read two numbers A and B and to print whether A is larger than, smaller than or equal to B using ternary operator.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int A, B;
        printf("Enter two numbers for A  and B \n");
        scanf("%d %d" , &A, &B);
        (A > B) ? printf("%d is Larger than %d\n", A, B) : ( A < B ? printf("%d is
Smaller than %d\n", A, B) : printf("%d is Equal to %d\n", A, B) ) ;
        return 0;
}
```

## 22. WACP to find the roots of a quadratic equation using else…if ladder or cascaded if statement.

**PROGRAM:**

```
#include<stdio.h>
#include<math.h>
int main()
{
        float a, b, c;
        float r1, r2, d;
        printf("Enter the coefficients a,b, and c:\n");
        scanf("%f%f%f", &a, &b, &c);
        if (a!=0 && b!=0 && c!=0)
        {
                d=(b*b)-(4*a*c);
                if (d>0)
                {
                        printf("roots are real and distinct :\n");
                        r1 = (-b+sqrt(d))/(2*a);
                        r2 = (-b-sqrt(d))/(2*a);
                        printf("root1=%f \n root2=%f\n", r1, r2);
                }
                else if (d<0)
                {
                        printf ("roots are imaginary : \n");
                        r1 = -b/(2*a);
                        r2 = sqrt(fabs(d))/(2*a);
                        printf("root1=%f + i %f \n root2=%f – i %f\n", r1, r2, r1, r2);
                }
                else
                {
                        printf ("roots are real and equal: \n");
                        r1 = -b/(2*a);
                        r2 = r1;
                        printf("root1=%f \n  root2= %f \n", r1, r2);
                }
        }
        else
                printf("All coefficients must be non zero \n");
        return 0;
}
```

## 23. WACP to print grade of a student based on marks obtained using else…if ladder statement.

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int marks;
        char grade;
        printf("Enter the marks obtained by a student\n");
        scanf("%d", &marks);
        if ( marks >= 90 )
                grade='S' ;
        else if ( marks >= 80 )
                grade='A' ;
        else if ( marks >= 70 )
                grade='B' ;
        else if ( marks >= 60 )
                grade='C' ;
        else if ( marks >= 50 )
                grade='D' ;
        else if ( marks >= 40 )
                grade='E' ;
        else
                grade='F' ;
        printf("Grade of a Student = %c \n", grade);
        return 0;
}
```

**24. WACP to print grade of a student based on marks obtained using switch statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int marks;
        char grade;
        printf("Enter the marks obtained by a student\n");
        scanf("%d", &marks);
        marks = marks / 10;
        switch (marks)
        {
                case 10:
                case 9: grade = 'S';
                        break;
                case 8: grade = 'A';
                        break;
                case 7: grade = 'B';
                        break;
                case 6: grade = 'C';
                        break;
                case 5: grade = 'D';
```

```
                                        break;
                        case 4: grade = ‘E’;
                                        break;
                        default: grade  = ‘F’;
            }
            printf(“Grade of a Student = %c \n”, grade);
            return 0;
}
```

**25. WACP to implement a basic arithmetic calculator using else…if ladder statement.**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int op1, op2;
        float result;
        char opt;
        printf(“Enter the expression to be evaluated \n”);
        scanf(“%d %c %d”, &op1, &opt, &op2);
        if ( opt = = ‘+’)
                result = op1 + op2;
        else if ( opt = = ‘–’)
                result = op1 – op2;
        else if ( opt = = ‘*’)
                result = op1 * op2;
        else if ( opt = = ‘%’)
                result = op1 % op2;
        else if ( opt = = ‘/’)
        {
                if ( op2 != 0 )
                        result = (float) op1 / op2;
                else
                {
                        printf(“ERROR: Divide by ZERO\n”);
                        return -1;
                }
        }
        else
        {
                printf(“Invalid Operator \n”);
                return -1;
        }
        printf(“Result of %d %c %d = %f \n”, op1, opt, op2, result);
        return 0;
}
```

## 26. WACP to implement a basic arithmetic calculator using switch statement.

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        int op1, op2;
        float result;
        char opt;
        printf("Enter the expression to be evaluated \n");
        scanf("%d %c %d", &op1, &opt, &op2);
        switch ( opt )
        {
                case '+':       result = op1 + op2;
                                break;
                 case '-':      result = op1 - op2;
                                break;
                case '*':       result = op1 * op2;
                                break;
                case '%':       result = op1 % op2;
                                break;
                case '/':       if ( op2 != 0 )
                                {
                                        result = op1 / op2;
                                        break;
                                }

                default: printf("ERROR: Divide by ZERO/Invalid Operator \n");
                         return -1;
        }
        printf("Result of %d %c %d = %f \n", op1, opt, op2, result);
        return 0;
}
```

## 27. WACP to determine a given character is VOWEL or not using else…if ladder statement.

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        char ch;
        printf("Enter a character \n");
        scanf("%c", &ch);
        if ( ch = = 'a' || ch = = 'A' )
                printf("%c is Vowel \n", ch);
        else if if ( ch = = 'e' || ch = = 'E' )
                printf("%c is Vowel \n", ch);
        else if if ( ch = = 'i' || ch = = 'I' )
                printf("%c is Vowel \n", ch);
        else if if ( ch = = 'o' || ch = = 'O' )
                printf("%c is Vowel \n", ch);
        else if if ( ch = = 'u' || ch = = 'U' )
                printf("%c is Vowel \n", ch);
        else
                printf("%c is not a Vowel \n", ch);
        return 0;
}
```

## 28. WACP to determine a given character is VOWEL or not using switch statement.

**PROGRAM:**

```
#include<stdio.h>
int main()
{
        char ch;
        printf("Enter a character \n");
        scanf("%c", &ch);
        switch ( ch )
        {
                case 'a' :
                case 'A':       printf("%c is a VOWEL \n",ch);
                                break;
                case 'e' :
                case 'E':       printf("%c is a VOWEL \n",ch);
                                break;
                case 'i' :
                case 'I':       printf("%c is a VOWEL \n",ch);
                                break;
                case 'o' :
                case 'O':       printf("%c is a VOWEL \n",ch);
                                break;
                case 'u' :
                case 'U':       printf("%c is a VOWEL \n",ch);
                                break;
```

```
                    default:        printf("%c is not a VOWEL\n" , ch);
            }
            return 0;
    }
```

## ii. Unconditional Branching Statements

The statements that help us to jump from one statement to another statement based on condition.

### a. goto statement

goto statement can be used to branch unconditionally from one point to another point in the program. The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name and must be followed by a colon ( : ). The label is placed immediately before the statement where the control is to be transferred.

The **syntax** is shown below:

```
            goto label1;
            statement1;
            statement2;
            statement3;
   label1 : statement4;
            statement5;
        Example 1
```

```
label2 :  statement1;
          statement2;
          statement3;
          goto label2;
          statement4;
          statement5;
    Example 2
```

In example 1, since the first statement is **goto label1** it jumps to label1 and starts executing the statement4 as shown with arrows. Hence the jump is made in forwarding direction.

In example 2, it first executes statements 1, 2 and 3 in sequence then because of the statement **goto label2,** it jumps to label2 and starts executing the statements 1, 2, and 3 and repeats the same process as shown with arrows. Hence the jump is made in backward direction and forms a loop.

From the example 2, we can understand one of the main drawback of using **goto** statement is that it forms infinite loop (i.e., loop that never ends) since it is unconditional jump statement.

Another drawback of using goto statement is that it is used to transfer the control only within the function but cannot be used to jump between functions.

**NOTE: Try to avoid using goto statement. It's a bad practice of using goto in a program.**

**Example**: Program to detect the entered number is even or odd using goto statement.

```
#include<stdio.h>
int main()
{
        int x;
        printf("Enter a Number: \n");
        scanf("%d", &x);
        if(x % 2 = = 0)
                goto even;
        else
                goto odd;
even :  printf("%d is Even Number" , x);
        return 0;
odd :   printf(" %d is Odd Number" , x);
        return 0;
}
```

**Output 1:**

```
Enter a Number: 5
5 is Odd Number.
```

**Output 2:**

```
Enter a Number: 8
8 is Even Number.
```

b. **break statement**

**The break statement can be used in any looping statements and switch statement.**

When a break statement is encountered inside a while, for, do…while statement, then break statement immediately terminates the entire loop and jumps out of the loop and resumes execution at the next statement following the loop (if any).

When a break statement is encountered inside a switch statement case, then break statement will terminate the corresponding switch case and transfers the control out of switch statement and resumes the execution at the next statement following the switch statement (if any).

Example 1:-

```
...
for ( x = 1 ; x <= 10  ; x++  )
{
        if ( x > 4  )
                break ;
        printf( "%d  " , x ) ;
}
printf( "Next Statement\n" );
```

The output of the above example will be : 1  2  3  4  Next Statement.

The reason for this output is that the for loop will iterate for values of x=1, 2, 3, 4 successfully and prints 1, 2, 3, 4 on screen since these values are lesser than 4 but when value of x becomes 5 then the condition within if statement i.e. x > 4 becomes true then it executes the break statement that terminates the remaining iterations of for loop and transfers control out of loop and continues with the next statement after for loop i.e. it executes printf statement. Therefore it prints 1 2 3 4 Next Statement as output.

Example 2:-

```
switch (ch)
{
        case 1: s1;
                break;
        ………..
        ………..
}
Next-Statement;
```

In the above example, if value of ch matches with case 1 then it executes statement s1 followed by break statement. After executing break statement the control is transferred to out of switch statement i.e. Next-Statement.

c. **continue statement**

**The continue statement can be used only in looping statements.**

The continue statement terminates the current iteration of a while, for or do…while statement and resumes execution of next iteration of the loop.

For Example :-

```
...
for ( x = 1; x <= 5; x++ )
{
        if ( x = = 3 )
                continue ;
        printf( "%d  ", x ) ;
}
```

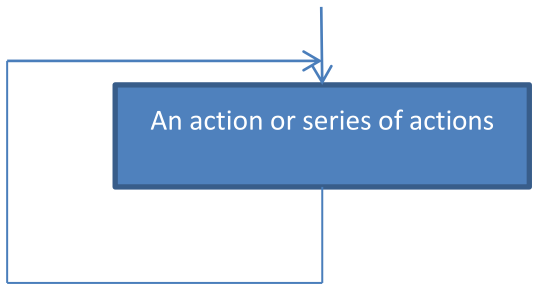
The output of the above example will be: 1  2  4  5.

The reason for this output is that the for loop will iterate for values of x=1, 2, 3, 4, and 5 but when the value of x is 3 then the condition within if statement i.e. x = = 3 becomes TRUE then it executes the continue statement that terminates the current iteration of for loop (skips following printf statement) and transfers control to the beginning of next iteration of for loop i.e. x++ and continues with the remaining iterations of for loop as shown with arrow. Therefore it prints 1 2 4 5 as output.

## Looping statements:

The statements that help us to execute set of statements repeatedly are called as **looping statements**.

The concept of loop is shown in the following flowchart.



In this flowchart, the action is repeated over and over again. It never stops.

Since the loop never stops, the action or actions will be repeated forever. We don't want this to happen; we want our loop to end when the work is done. To make sure that it ends, we must have a condition that controls the loop. The condition which is used to control the loop is called as loop control expression.

Based on the placement of loop control expression, the loops are classified as TWO types

i. **Entry controlled loop or Pre-Test Loop:**

In the entry controlled loop or Pre-Test loop, the loop control expression is checked before we start and at the beginning of each iteration of the loop. If the control expression is true, we execute action or actions; if the control expression is false, we terminate the loop.

Examples:

while statement and for statement

ii. **Exit controlled loop or Post-Test Loop**

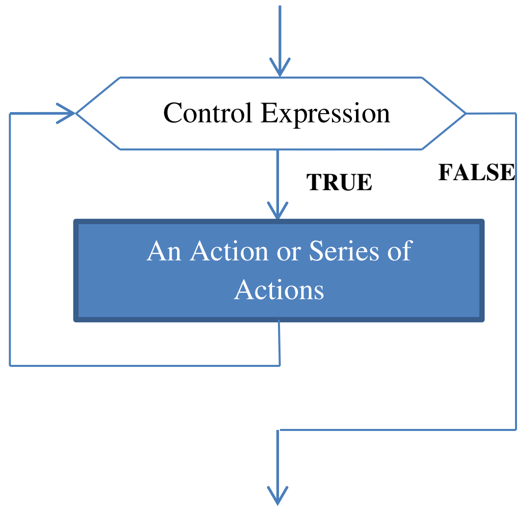
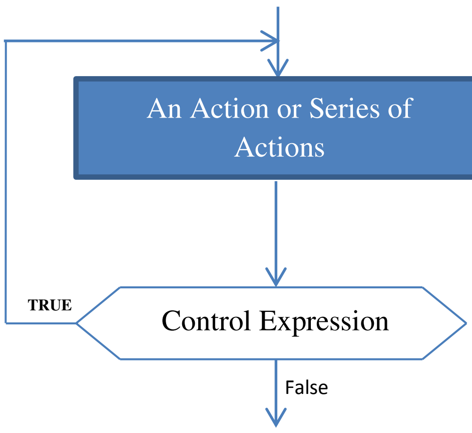
In the exit controlled loop or Post-Test loop, the loop control expression is checked after or at the end of each iteration of the loop. If the control expression is true, we repeat action or actions; if the control expression is false, we terminate the loop.

Examples:

do…while statement

The flowchart of entry controlled and exit controlled loops and other differences between the two are given below.

| **Entry Controlled or Pre-Test Loop** | **Exit Controlled or Post-Test Loop** |
|---|---|
| The loop control expression is checked before we start and at the beginning of each iteration of the loop | The loop control expression is checked after or at the end of each iteration of the loop |

| | |
|---|---|
| Flowchart:   Control Expression<br>TRUE<br>FALSE<br>An Action or Series of Actions | Flowchart:   An Action or Series of Actions<br>TRUE<br>Control Expression<br>False |
| Minimum Iterations: Atleast ZERO Times | Minimum Iterations: Atleast ONE Times |
| If the control expression is TRUE for N times, then action or actions are executed for N times | If the control expression is TRUE for N times, then action or actions are executed for N+1 times |
| Examples: for loop and while loop | Example: do…while loop |
| Prints numbers from 1 to 10<br>i=1;<br>while( i<=10)<br>{<br>    printf("%d\n", i);<br>    i++;<br>} | Prints numbers from 1 to 10<br>i=1;<br>do<br>{<br>    printf("%d\n", i);<br>    i++;<br>}while(i<10); |

**Initialization:** Before a loop can start, some preparation is usually required. We call this preparation as initialization. Initialization must be done before the execution of the loop body. Initialization sets the variable which will be used in control expression to control the loop.

**Updating:** how can the control expression that controls the loop be true for a while and then change to false? The answer is that something must happen **inside** the body of the loop to change the control expression. Otherwise, the loop leads to infinite loop. Hence, the actions that cause these changes are known as **update.**

### a. while statement

The while statement is typically used in situations where it is not known in advance how many iterations are required.

Syntax:

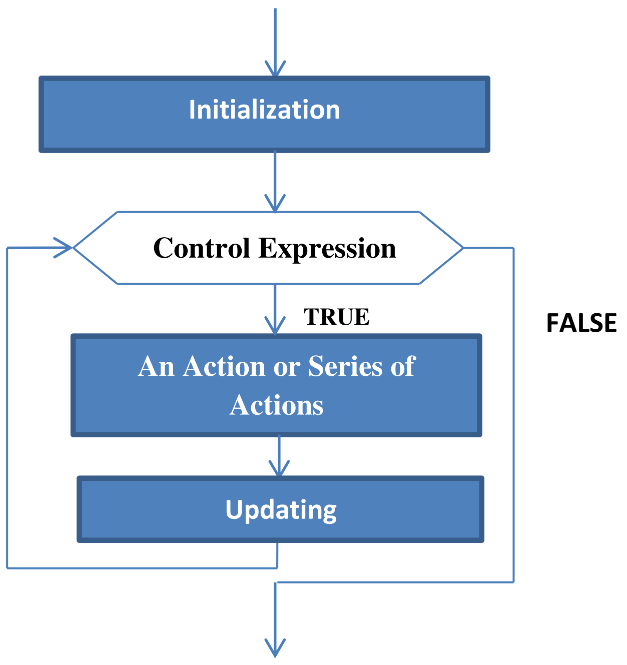
```
while ( control expression )
{
        An action or Series of Actions ;        //body of loop
}
```

Here, first the control expression is evaluated to TRUE or FALSE. If it is evaluated to TRUE, then an action or series of action i.e. body of loop is executed. The process of executing the body of loop is repeated until the control expression is FALSE. Once control expression is evaluated to FALSE, then the control is transferred out of the loop. The execution resumes from the statement following the loop (if any).

Generally, when initialization and updating is included in the while loop, it looks as follows.

```
        Initialization          // initialization is used before the start of loop and sets
                                // the variable(s) to control the loop
        while ( control expression )
        {
                An action or Series of Actions ;        //body of loop
                Updating        // causes the variable(s) to update that helps to repeat
                                // the loop or to stop the loop
        }
```

The flow of while statement is shown below.



**Example**: Program to sum all integers from 1 to 100.

```
#include <stdio.h>
int main()
{
        int sum = 0, i;
        i=1;
        while ( i<=100 )
        {
                sum = sum + i;
                i++;
        }
        printf( “Sum = %d \n”, sum ) ;
        return 0;
}
```

In this example, it will repeat the statements sum=sum+i and i++ till the control expression i.e. i<=100 is **TRUE**. Once the control expression is evaluated to **FALSE**, it goes out of the loop and then executes the printf statement.

## b. for statement

The for statement is most often used in situations where the programmer knows in advance how many times a particular set of statements are to be repeated. The for statement is sometimes termed as counter controlled loop.

Syntax :

```
                for ( expression1 ; expression2 ; expression3 )
                {
                        An action or series of actions ;        // body of loop
                }
```

Where,

**expression1**:- This is usually an assignment/initialization statement to set a loop control variable(s) for example. But it is not restricted to only initialization, it can be any valid C statement.

**expression2**:- This is usually a control expression which determines when loop will terminate. But, this can be any statement of C that evaluates to TRUE or FALSE.

**expression3**:- This usually defines how the loop control variable(s) will change each time the loop is executed i.e. updating.

**Body of loop**:- Can be a single statement, no statement or a block of statements.

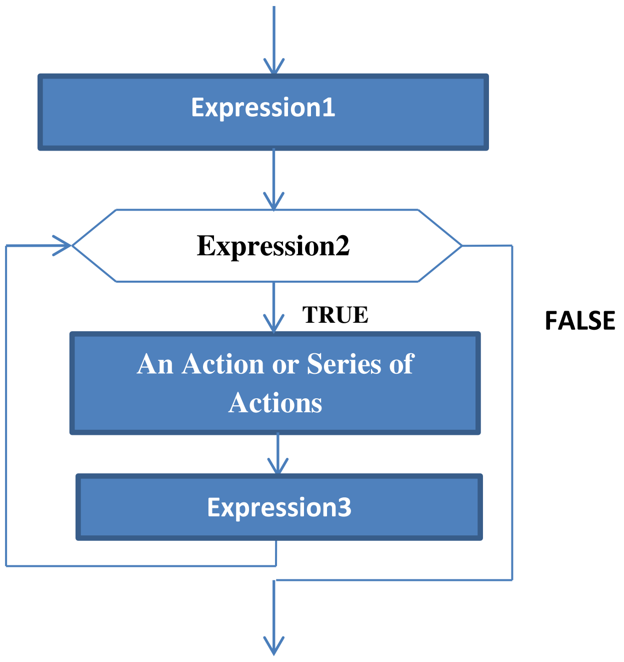
**NOTE**: **All three expressions are optional. But semicolons must be present as in syntax in the loop.**

Curly braces are used in C to denote block whether in a function as in main() or as the body of a loop. It is optional if body of loop is a single statement or no statement.

The for statement executes as follows:

Example 1: Program to sum all numbers from 1 to 100.

```
 #include <stdio.h>
 int main()
{
        int x, sum=0 ;
        for ( x = 1;  x <= 100; x++ )
                sum = sum + x;
        printf(“Sum = %d\n”, sum);
        return 0;
}
```



Example 2:- To print out all numbers from 1 to 100.

```
#include <stdio.h>
int main()
{
        int x;
        for ( x = 1; x <= 100; x++ )
                printf( "%d\n", x ) ;
        return 0;
}
```

**Multiple Initialisations in for loop**

C has a special operator called the comma operator which allows separate expressions to be tied together into one statement.

For example it may be tidier to initialise two variables in a for loop as follows:-

```
                for ( x = 0, sum = 0; x <= 100; x++ )
                {
                        printf( "%d\n", x) ;
                        sum += x ;
                }
```

Any of the four sections associated with a for loop may be omitted but the semi-colons must be present always.

For Example 1:- expression 3 is omitted

```
                for ( x = 0; x < 10;   )
                        printf( "%d\n", x++ ) ;
```

For Example 2:- expression 1 is omitted in for loop and it is before for loop

```
                x = 0 ;
                for (  ; x < 10; x++ )
```

```
                    printf( "%d\n", x ) ;
```

For Example 3:- An infinite loop may be created as follows

```
          for ( ;  ;  )
                    body of loop;
```

For Example 4:- Sometimes a for statement may not even have a body to execute as in the following example where we just want to create a time delay.

```
          for ( t = 0; t < big_num ; t++ )
                    ;
```

For Example 5:- expression 3 is a printf statement. This example prints from 1 to 100.

```
          for ( x = 1; x <= 100; printf( "%d\n", x++ ) ) ;
```

For Example 6:- The expression1, expression2 and expression3 sections of the for statement can contain any valid C expressions.

```
          for ( x = 12 * 4 ; x < 34 / 2 * 47 ; x += 10 )
                    printf( "%d ", x ) ;
```

Nested for loops:- It is possible to build a nested structure of for loops, for example the following creates a large time delay using just integer variables.

```
     unsigned int x, y ;
     for ( x = 0; x < m; x++ )
          for ( y = 0; y < n; y++ )
                    ;
```

In this example, the outer for loop will iterate for ‘m’ times and inner loop will iterate for ‘n’ times, hence the total iterations from nested loop is **m*n times**.

For Example: Program to produce the following table of values

```
1    2    3    4    5
2    3    4    5    6
3    4    5    6    7
4    5    6    7    8
5    6    7    8    9
```

```
#include <stdio.h>
int main()
{
     int j, k ;
     for ( j = 1; j <= 5; j++ )
     {
          for ( k = j ; k < j + 5; k++ )
          {
               printf( "%d  ", k ) ;
          }
          printf( "\n" ) ;
     }
     return 0;
}
```

## c. do…while statement

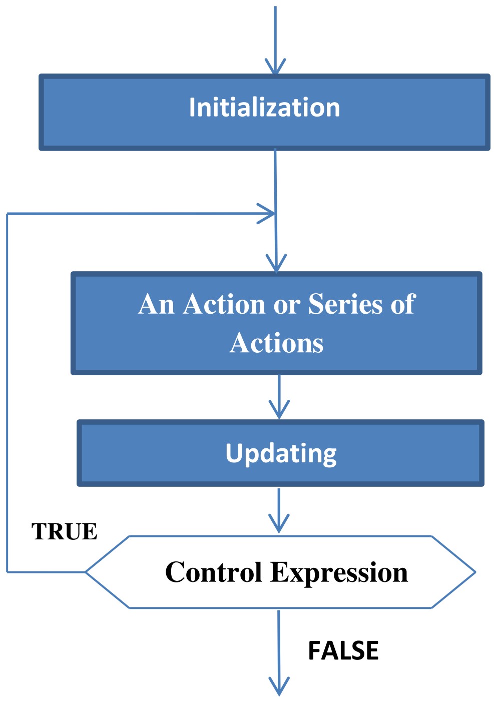
The terminating condition in the for and while loops is always tested before the body of the loop is executed -- so of course the body of the loop may not be executed at all.

In the do…while statement on the other hand the body of the loop is always executed at least once as the condition is tested at the end of the body of the loop.

Syntax :

```
Initialization
do
{
        An action or Series of Actions ;        // body of loop
        Updating
}  while ( control expression ) ;
```

The flow of do…while loop is shown below



For Example : To sum all numbers from 1 to 100.

```
int i, sum=0;
i = 1;
do
{
        sum = sum + i ;
        i++;
} while ( i <= 100 ) ;
```

## Programming examples on Looping:

**1. WACP to find the sum of numbers from 1 to N where N is entered from keyboard.**

**Program:**

**// sum of numbers from 1 to N i.e. sum = 1+2+3+4+5+…+N**

```
#include<stdio.h>
int main()
{
    int N, sum=0, i;
    printf("Enter the value of N\n");
    scanf("%d" , &N);
    i=1;
    while(i<=N)
    {
            sum = sum + i;
            i++;
    }
    printf("Sum = %d\n" , sum);
    return 0;
}
```

**2. WACP to find the sum squares of numbers from 1 to N where N is entered from keyboard.**

**Program:**

**// sum of squares of numbers from 1 to N i.e. sum = $1^2+2^2+3^2+4^2+5^2+…+N^2$**

```
#include<stdio.h>
#include<math.h>
int main()
{
        int N, sum=0, i;
        printf("Enter the value of N\n");
        scanf("%d" , &N);
        i=1;
        while(i<=N)
        {
                sum = sum + pow(i,2);
                i++;
        }
        printf("Sum = %d\n" , sum);
        return 0;
}
```

**3. WACP to evaluate the following f(x) = 1 + X + $X^2$ + $X^3$ + … + $X^N$, where X and N are entered from keyboard.**

**Program:**

**// f(x) = 1 + X + $X^2$ + $X^3$ + … + $X^N$**

```
#include<stdio.h>
#include<math.h>
int main()
{
        int N, X, sum=0, i;
        printf("Enter the value of X and N\n");
        scanf("%d%d" , &X, &N);
        for(i=0; i<=N; i++)
                sum = sum + pow(X,i);
        printf("Sum = %d\n" , sum);
        return 0;
}
```

**4. WACP to find and print reverse number of a given integer number.**

**Program:**

```
#include<stdio.h>
int main()
{
        int N, reverse=0,rem,temp;
        printf("Enter the value of N\n");
        scanf("%d" , &N);
        temp=N;
        while(N>0)
        {
                rem = N%10;
                reverse = reverse * 10 + rem;
                N = N/10;
        }
        printf("Given Number = %d\n", temp);
        printf("Reversed Number = %d\n", reverse);
        return 0;
}
```

**5. WACP to find reverse number of a given integer number and print whether it is a palindrome or not.**

**Program:**

```
#include<stdio.h>
int main()
{
        int N, reverse=0,rem, temp;
        printf("Enter the value of N\n");
        scanf("%d" , &N);
        temp = N;
        while(N>0)
        {
                rem = N%10;
                reverse = reverse * 10 + rem;
                N = N/10;
        }
        printf("Given Number = %d\n", N);
        printf("Reversed Number = %d\n", reverse);
        if(N = = temp)
                printf("%d is Palindrome\n", temp);
        else
                printf("%d is not Palindrome\n", temp);
        return 0;
}
```

**6. WACP to find sum of digits of a given integer number. ( For example, 123 = 1 + 2 + 3 = 6).**

**Program:**

```
#include<stdio.h>
int main()
{
        int N, sum=0, rem;
        printf("Enter the value of N\n");
        scanf("%d" , &N);
        while (N>0)
        {
                rem = N%10;
                sum = sum + rem;
                N = N/10;
        }
        printf(" Sum of Digits = %d\n", sum);
        return 0;
}
```

## 7. WACP to find the factorial of a given number.

**Program:**

```
#include<stdio.h>
int main()
{
        int N, fact=1, i;
        printf("Enter the value of N\n");
        scanf("%d" , &N);
        i=1;
        do
        {
                fact = fact * i;
                i++;
        } while(i<=N);
        printf("Factorial(%d) = %d\n", N, fact);
        return 0;
}
```

## 8. WACP to find the GCD and LCM of given integer numbers.

**Program:**

```
#include<stdio.h>
int main()
{
        int M, N, P,Q, GCD, rem, LCM;
        printf("Enter the value of M and N\n");
        scanf("%d%d" , &M, &N);
        P=M;
        Q=N;
        while (N>0)
        {
                rem = M%N;
                M=N;
                N = rem;
        }
        GCD=M;
        printf(" GCD ( %d, %d ) = %d\n", P, Q, GCD);
        LCM = (P* Q) / GCD;
        printf(" LCM ( %d, %d ) = %d\n", P, Q, LCM);
        return 0;
}
```

## 9. WACP to find the sum of even and odd numbers from 1 to N.

**Program:**

```
#include<stdio.h>
int main()
{
        int N, i, even_sum=0, odd_sum=0;
        printf("Enter the value of N\n");
        scanf("%d" , &N);
        for(i=1; i<=N; i++)
        {
                if( i % 2 = = 0)
                        even_sum += i;
                else
                        odd_sum += i;
        }
        printf(" Sum of Even Numbers = %d\n", even_sum);
        printf(" Sum of Odd Numbers = %d\n", odd_sum);
        return 0;
}
```

## 10. WACP to print alternate uppercase letters starting with letter 'A'.

**Program:**

```
#include<stdio.h>
int main()
{
        char ch;
        printf("Alternate Uppercase letters starting with letter A are:\n");
        for(ch='A'; ch<='Z'; ch= ch + 2)
            printf("%c\n", ch);
        return 0;
}
```

## 11. WACP to find the GCD of two numbers using while and ternary operator.

**Program:**

```
#include<stdio.h>
int main()
{
        int M, N, GCD, P,Q;
        printf("Enter the value of M and N\n");
        scanf("%d%d" , &M, &N);
        P=M;
        Q=N;
        while (M != N)
                (M>N) ? M = M – N : N = N – M ;
        GCD=M;
        printf(" GCD ( %d, %d) = %d\n", P,Q, GCD);
        return 0;
}
```

**12. WACP to find the GCD of two numbers using while and if else statement.**

**Program:**

```
#include<stdio.h>
int main()
{
        int M, N, GCD, P,Q;
        printf("Enter the value of M and N\n");
        scanf("%d%d" , &M, &N);
        P=M;
        Q=N;
        while ( M != N )
        {
                if (M>N)
                        M = M – N;
                else
                         N = N – M ;
        }
        GCD=M;
        printf(" GCD ( %d, %d) = %d\n", P, Q, GCD);
        return 0;
}
```

**13. WACP to find a given number is prime or not.**

**Prime number: A number which is divisible by 1 and itself.**

**Program:**

```
#include<stdio.h>
int main()
{
        int N, i;
        printf("Enter the value of N\n");
        scanf("%d" , &N);
        for(i=2; i<=N/2; i++)
        {
                if( N % i = = 0)
                {
                        printf("%d is not a prime number\n", N);
                        return 0;
                }
        }
        printf("%d is a prime number\n", N);
        return 0;
}
```

*****ALL THE BEST*****