

**K.S. SCHOOL OF ENGINEERING AND MANAGEMENT,  
BANGALORE - 560109**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Affiliated to  
Visvesvaraya Technological University**



**Lab Manual**

## **INTRODUCTION TO C PROGRAMMING**

**(Subject Code: BESCK104E/204E)**



**Prepared by:  
Ms. Ambuja K  
Assistant Professor  
Dept. of CSE, KSSEM**

**Mrs. Prasanna N  
Assistant Professor  
Dept. of CSE, KSSEM**

**Department of Computer Science and Engineering**

**Raghuvanahalli, Kanakapura Main Road,  
Bangalore 560109**

## INTRODUCTION

Numerous examples in the past where human beings have used either living or nonliving objects as tools to overcome inherent disabilities. Computers are just one of the latest tools being designed and used to overcome inherent disability.

The main disability of human beings is that most of us cannot perform large and complicated computations accurately and quickly. So designed a tool called the computer which can perform computations with a very high accuracy and speed. However computers will have to be given instructions to perform computation. **Set of instructions given to the computer to perform computations is technically called a program.**

The programmer cannot start to write a program directly. He has to plan before writing a program. This is somewhat similar to the fact that if a person decides to construct a house, the very next day, he would not start to construct the house directly. He has to plan each and every minor detail before proceeding to construct the house. Once the entire plan (sketch) of the house is ready, then he can proceed and construct the house.

Similarly, before writing a program, the programmer has to plan the steps and the sequence in which these steps must be performed to obtain the correct results. If the programmer skips (forget) a few steps or perform the steps in a wrong sequence, then the computer would produce a wrong result.

Hence, programs must be planned before they are written. During this process of planning the programmer must first try to write an algorithm, then a flow chart, and then try to convert the flow chart to a program.

So the sequence of steps involved in the problem solving is as follows:

1. Understand the problem.
2. Express the solution to the problem in the form of an **algorithm**.

3. Express the algorithm pictorially in the form of **flow chart**
4. Express the flow chart in the form of a program
5. Feed the program to a computer
6. Obtain the result

Algorithm and flowcharts are the initial preparatory steps taken by the programmer before writing the actual program. First writing an algorithm then a flow chart to a problem and then writing a program to that problem makes programming easier, rather than directly writing a program.

## **ALGORITHM**

Algorithm is defined as a sequence of unambiguous instructions designed in such a way that if the instructions are performed in the specified sequence, the desired results would be obtained.

However it is important to note that an algorithm cannot be directly fed into the computer for its execution. An algorithm just represents the logic to solve a problem in a step by step manner.

### **How to write an algorithm?**

If the algorithm has to follow the following notation:

- The name of the algorithm must be specified.
- Step number must be given to each of the steps for the sake of identification of the steps.
- Each step must have an explanatory note provided within the square brackets followed by the corresponding operation.
- The completion of an algorithm must be specified towards the end, using a stop statement

The above steps would become clear with the following example

Write an algorithm to compute simple interest.

Logic: Simple interest can be computed using the formula  $si = p * t * r / 100$ .

Algorithm: Simple interest

Step 1: [input principle, time and rate of interest]

Read p,t and r

Step 2: [Compute the simple interest]

$Si=(p*t*r)/100;$

Step 3: [output the result]

print si

Step 4: [Finished]

Stop

## **Purpose for writing an algorithm**

- **Effective communication:** Since an algorithm is written using English like statements, it is more readable and understandable.
- **Effective analysis:** Writing an algorithm would help us to obtain an in-depth understanding of the problem and hence would enable us to write programs easily.
- **Effective coding:** Once an algorithm is ready, programmers find it very easy to write the corresponding program because an algorithm acts as a road map for them.
- **Effective debugging:** Once an algorithm is ready, we can identify if there are any errors and debug (rectify) such errors.
- **Easy maintenance:** Algorithm also provides for proper documentation and hence supports easy maintenance of the software.

## **Flowcharts**

A flow chart is a pictorial representation of an algorithm that uses boxes of different shapes to denote the flow of the control.

Normally, algorithms are not directly converted to programs. An algorithm is first represented in the form of a flow chart and the flow chart is then used to write a program.

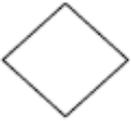
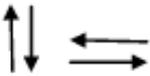
Flow chart shows the flow of control in a pictorial form, any errors in the logic of the program can be easily detected.

It is important to note that the experienced programmers sometimes write program directly without drawing the flow chart. However for a beginner, it is always recommended to write an algorithm and a flow chart and then only write a program in order to reduce the

number of errors in the program and also make programming easier.

## Geometric figures used in flow chart

A flow chart uses different geometric shape to denote different types of actions. Some of the standard geometric shapes and their meaning are as given below

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

## SAMPLE PROGRAMS

### **Program1: Write a C program for Calculation of Simple Interest.**

**Description:** Simple interest is a method to calculate the amount of interest charged on a sum at a given rate and for a given period of time.

Simple interest is calculated with the following formula:  $S.I. = P \times R \times T$ , where P = Principal, R = Rate of Interest in % per annum, and T = Time, usually calculated as the number of years. The rate of interest is in percentage r% and is to be written as r/100.

### **Algorithm:**

**Step 1:** Start.

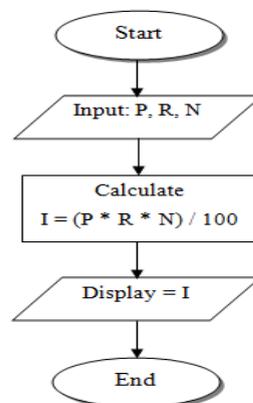
**Step 2:** Read Principal Amount, Rate and Time.

**Step 3:** Calculate Interest using formula  $SI = ((\text{amount} * \text{rate} * \text{time}) / 100)$

**Step 4:** Print Simple Interest.

**Step 5:** Stop.

### **Flow Chart:**



### **Source Code:**

```
#include<stdio.h>
int main()
{
    int p,r,t,int_amt;
    printf("Input principle, Rate of interest & time to find simple interest: \n");
    scanf("%d%d%d",&p,&r,&t);
    int_amt=(p*r*t)/100;
    printf("Simple interest = %d",int_amt);
    return 0;
}
```

**Input - Output:**

Enter the principal amount: 18000

Enter the Time Period: 2

Enter the Rate of Interest: 6

Simple interest for Principal Amount 18000.00 is 2160.00

**Program 2: Write a C program to check whether the given number is Even or Odd.**

**Description:** A number which is divisible by 2 and generates a remainder of 0 is called an even number. An odd number is a number which is not divisible by 2. The remainder in the case of an odd number is always “1”.

**Algorithm:**

**Step 1:** Start

**Step 2:** READ number

**Step 3:** remainder=number%2

**Step 4:** if remainder == 0

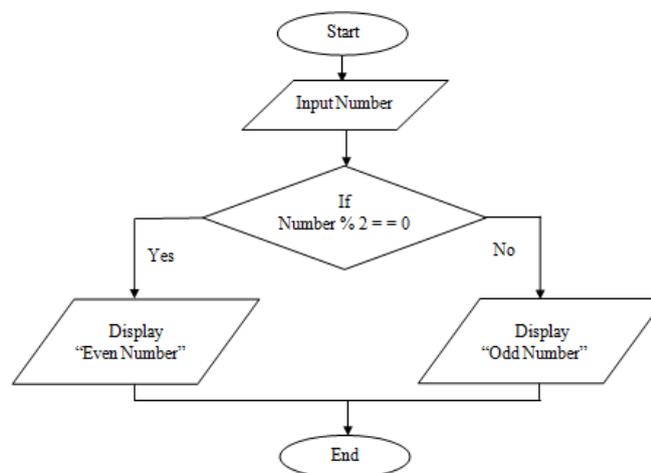
    Display "Even Number"

    else

    Display "Odd Number"

**Step 5:** Stop

**Flow Chart:**



**Source Code:**

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

**Input - Output:**

Enter the number: 2  
2 is even

Enter the number: 3  
3 is odd

**Program 3: Write a C program to convert String Case.**

**Description:** Given a string, convert the characters of the string into opposite case, i.e. if a character is lower case then convert it into upper case and vice-versa.

**Algorithm for lowercase to uppercase:**

**Step 1:** Start

**Step 2:** Check if the character is between 'a' and 'z' i.e. it is a lower case letter.

**Step 3:** If the character is a lower case letter, we subtract 32 from it.

**Step 4:** Else, the character is already in upper case. Do nothing.

**Step 5:** Stop

**Algorithm:**

**Algorithm to convert uppercase to lowercase:**

**Step 1:** Start

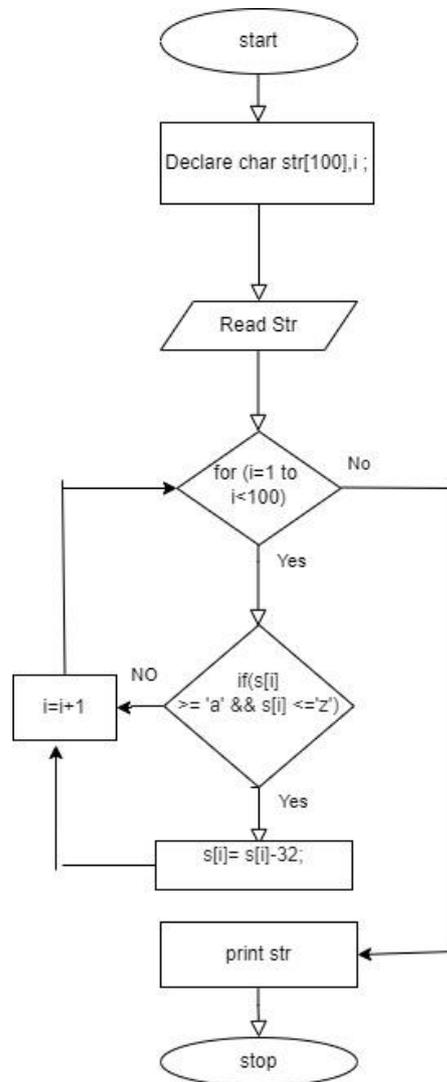
**Step 2:** Check if the character is between A and Z i.e. it is a capital letter,

**Step 3:** If the character is a capital, we add 32 to it.

**Step 4:** Else, the character is already in lower case. Do nothing.

**Step 5: Stop**

**Flow Chart:**



**Source Code:**

**Lowercase to Uppercase**

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[100];
int i;
printf("\n Enter a string : ");
gets(s);
for(i=0;s[i]!='\0'; i++)
{
if(s[i] >= 'a' && s[i] <= 'z')
{
s[i]= s[i]-32;
}
}
```

```
}  
printf(" String in Upper case : %s", s);  
return 0;  
}
```

**Input - Output:**

Enter a string: kssem  
String in Upper case : KSSEM

**Uppercase to Lowercase:**

```
#include<stdio.h>  
#include<string.h>  
int main()  
{  
char s[100];  
int i;  
printf("\n Enter a string : ");  
gets(s);  
for(i=0;s[i]!='\0'; i++)  
{  
if(s[i] >= 'A' && s[i] <='Z')  
{  
s[i]= s[i]+32;  
}  
}  
printf(" String in Lower case : %s", s);  
return 0;  
}
```

**Input - Output:**

Enter a string : KSSEM  
String in Lower case : kssem

**Program 4: Write a C program to check for the Palindrome.**

**Description:** A palindrome is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as madam or racecar.

**Algorithm:**

**Step 1:** Start

**Step 2:** Take a number num as input.

**Step 3:** Copy value of num to another variable say temp i.e. temp = num.

**Step 4:** Initialize a variable to 0 say reverse i.e. rev = 0.

**Step 5:** Perform rem=num%10.

**Step 6:** Perform num=num/10.

**Step 7:** Perform  $rev=rem+(rev*10)$ .

**Step 8:** if  $temp==rev$

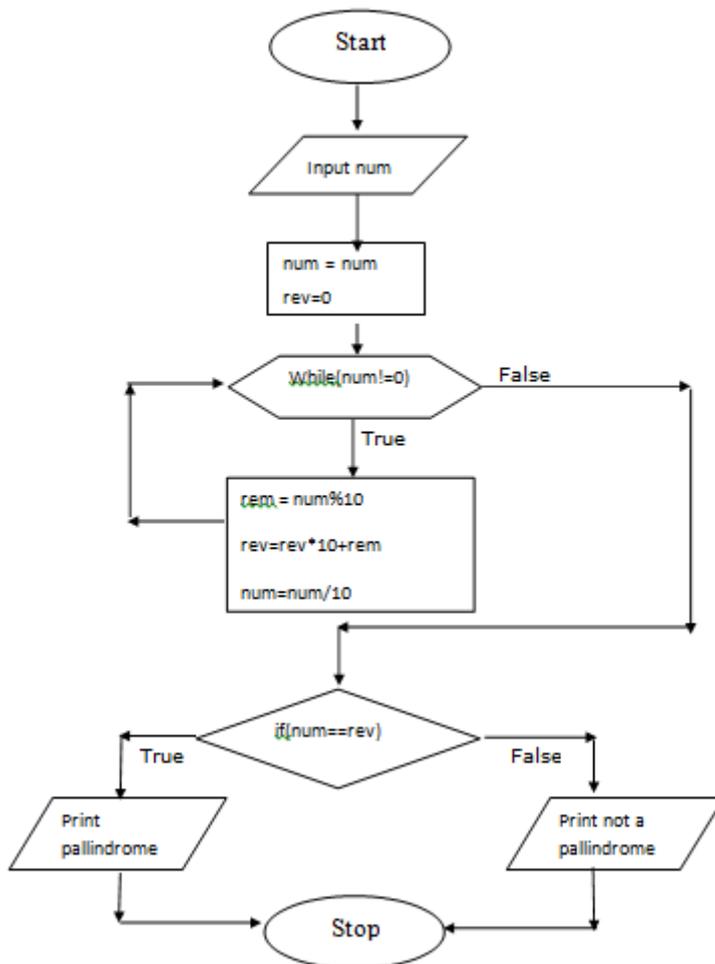
Display Palindrome.

Else

Not a palindrome

**Step 9:** Stop

**Flow Chart:**



**Source Code:**

```
#include<stdio.h>
int main()
{
int num,rem,rev,temp;
printf("Enter a positive integer number:\n");
scanf("%d", &num);
rev=0;
temp=num;
```

```
while(num!=0)
{
    rem=num%10;
    num=num/10;
    rev=rem+(rev*10);
}
if(temp==rev)
    printf("The given number %d is palindrome", temp);
else
    printf("The given number %d is not a palindrome", temp);
}
```

**Input - Output:**

Enter a positive integer number:

121

The given number 121 is palindrome

Enter a positive integer number:

456

The given number 456 is not a palindrome

**Program 5: Write a C program to check whether the given number is Prime or not.**

**Description:** Prime numbers are natural numbers that are divisible by only 1 and the number itself. In other words, prime numbers are positive integers greater than 1 with exactly two factors, 1 and the number itself. Some of the prime numbers include 2, 3, 5, 7, 11, 13, etc.

**Algorithm:**

Input: Any integer value

Output: The entered value is prime or not.

Step 1: Start

Step 2: Read value.

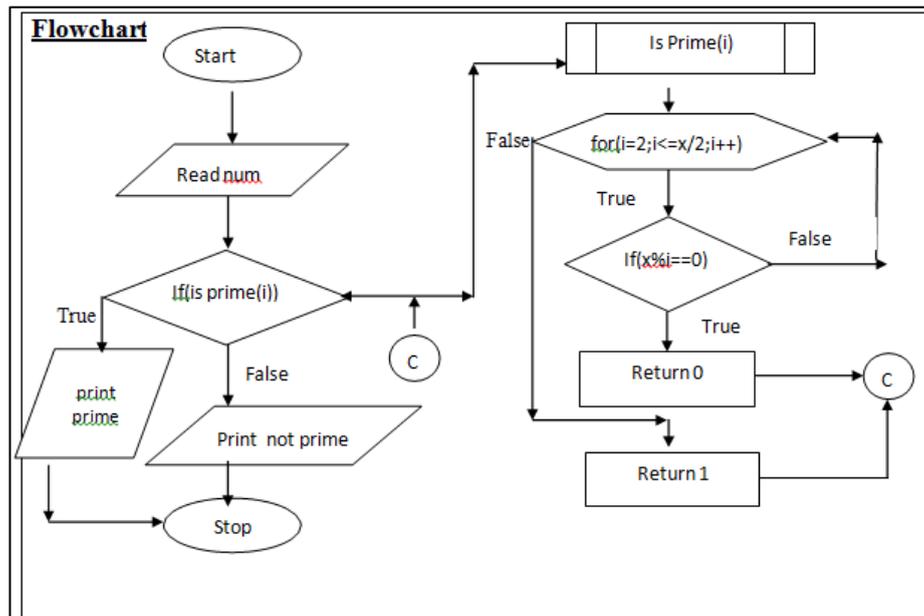
Step 3: By using for loop find the prime number between n1 and n2 using function.

Step 4: Using for loop, check whether the number is divisible or not.

Step 5: If divisible, terminate loop and return 0. If not divisible, terminate loop and return 1.

Step 6: If 1, display number is prime. If 0, display number is not prime.

Step 7: Stop.



**Source Code:**

```

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

return 1;
}
  
```

**Input - Output:**

Enter value of num:  
17  
17 is a prime number

Enter value of num:  
10  
10 is not a prime number

**Program 6: Write a C Program to check for perfect square.**

**Description:**

A perfect square is when the two equal numbers are multiplied by each other. For example, let us consider the number 25. 25 is a perfect square as we are multiplying two equal integers  $5*5$  by each other.

**Algorithm:**

**Step 1:** Start.

**Step 2:** Read variable i.

**Step 3:** Take a number from the user say x.

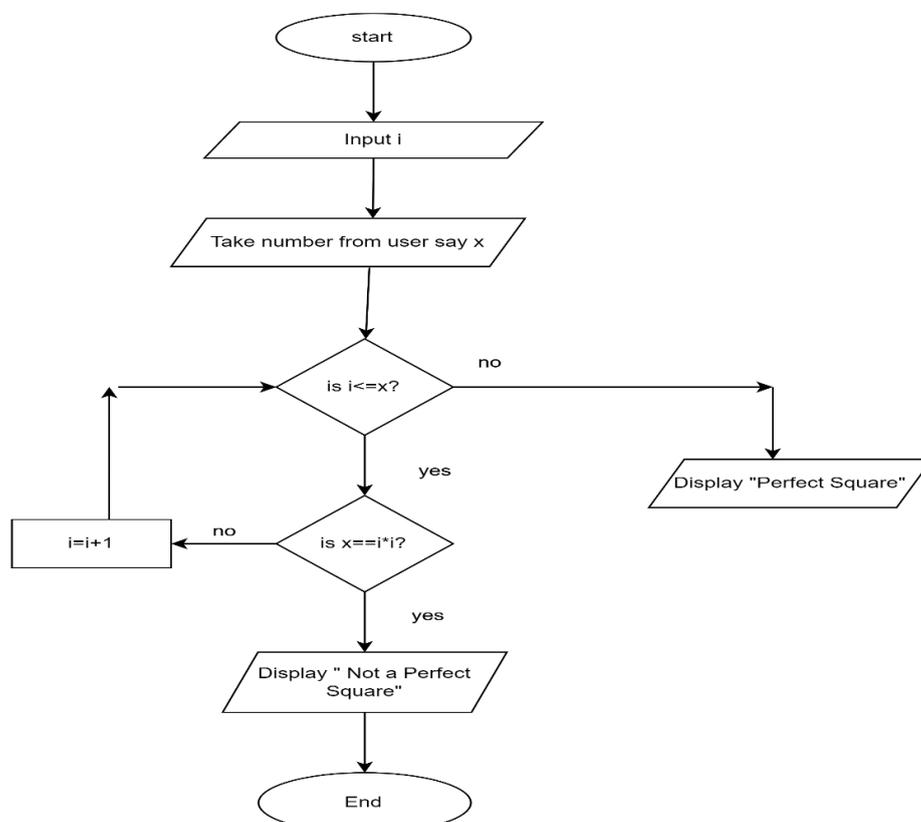
Step 4: If a variable  $i \leq 0$ , then if  $x = i*i$

Step 5: Display "Given Number is Perfect Square".

**Step 6:** Otherwise Display "Given Number is not a Perfect Square."

**Step 7:** Stop.

**Flow Chart:**



### **Source Code:**

```
#include<stdio.h>
int main()
{
    int i, x;

    printf("Enter a number: ");
    scanf("%d", &x);
    for(i = 0; i <= x; i++)
    {
        if(x == i*i)
        {
            printf("%d is a perfect square \n", x);
            return 0;
        }
    }
    printf("%d is not a perfect square\n", x);
    return 0;
}
```

### **Input - Output:**

Enter a number: 50  
50 is not a perfect square

Enter a number: 49  
49 is a perfect square

### **Program 7: Write a C program to demonstrate linear search algorithm.**

#### **Description:**

Linear search is also called as **sequential search algorithm**. It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

#### **Algorithm:**

Step 1: Start.

Step 2: Take an array input and let us name it as array[5].

Step 3: Declare **c**, **n** for number of elements in an array and **search** variable for elements to be searched.

Step 4: Linearly traverse the array using for loop.

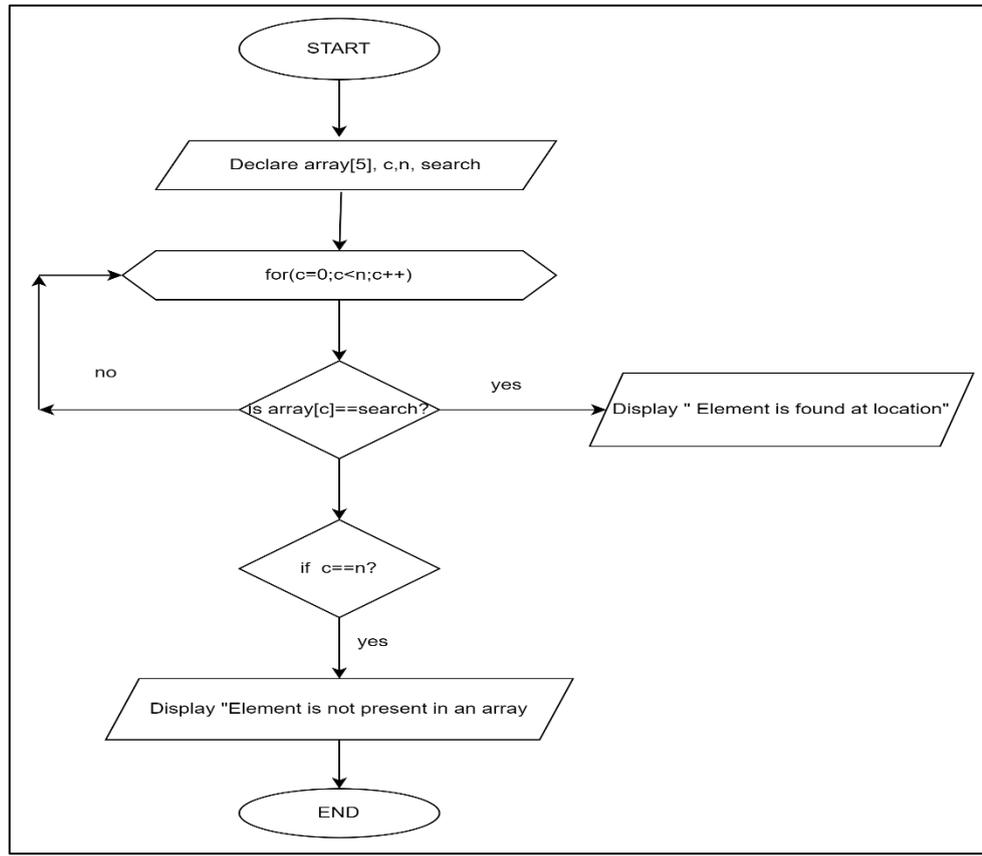
Step 5: For each element in an array, check if array[c]==search.

Step 6: If element is found in an array, then display “Number is present at location”.

Step 7: Check if  $c==n$ , then display “Element is not present in an array”.

Step 8: Stop.

**Flow Chart:**



**Source Code:**

```
#include <stdio.h>
int main()
{
    int array[5], search, c, n;

    printf("Enter number of elements in array\n");
    scanf("%d", &n);

    printf("Enter %d integer(s)\n", n);

    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);

    printf("Enter a number to search\n");
```

```
scanf("%d", &search);

for (c = 0; c < n; c++)
{
    if (array[c] == search) /* If required element is found */
    {
        printf("%d is present at location %d.\n", search, c+1);
        break;
    }
}
if (c == n)
    printf("%d isn't present in the array.\n", search);
return 0;
}
```

**OUTPUT:**

Enter number of elements in array

5

Enter 5 integer(s)

25 10 36 44 56

Enter a number to search

13

13 isn't present in the array.

Enter number of elements in array

5

Enter 5 integer(s)

10 20 30 40 50

Enter a number to search

40

40 is present at location 4.

**1. C Program to find Mechanical Energy of a particle using  $E = mgh + \frac{1}{2}mv^2$ .**

**ALGORITHM**

Step1: Start

Step 2: Initialize or declare variables m,v,h, K,E,P

Step3: Read mass m ,velocity v and displacement h from the user

Step3: calculate Potential energy  $P E=m*(9.8)*h$

Step4: calculate Kinetic energy  $K E= 0.5*m*v*v$

Step5: calculate Mechanical energy  $E= P.E+K.E$

Step6: Print the result

Step7: Stop

**SOURCE CODE**

```
#include <stdio.h>

int main( )
{
float m,h,v,p,k,e;
printf("Enter Mass of the body\n");
scanf("%f",&m );
printf("Enter displacement of the body\n");
scanf("%f",&h );
printf("Enter velocity of the body\n");
scanf("%f",&v );
p=m*9.8*h; //To calculate Potential energy
k=0.5*m*(v*v); //To calculate Kinetic energy
e=p+k;
printf("Potential energy of the body = %f\n",p );
printf("Kinetic energy of the body = %f\n",k );
printf("Mechanical energy of a body = %f\n", e);
}
```

**OUTPUT**

Enter Mass of the body

100

Enter displacement of the body

10

Enter velocity of the body

120

Potential energy of the body = 9800.000000

Kinetic energy of the body = 720000.000000

Mechanical energy of a body = 729800.000000

## 2. C Program to convert Kilometers into Meters and Centimeters.

### ALGORITHM

Step1: Start

Step2: Declare the variables km, cm, m.

Step3: Read the distance in Kilometers

Step4: Convert distance to meters,  $m = km * 1000.0$

Convert distance to centimeters,  $cm = km * 100000.0$

Step5: Print the result.

Step6: Stop

### Source code

```
#include<stdio.h>
int main( )
{
    float km, cm, m;
    printf("Enter distance in Kilometer\n");
    scanf("%f ", &km);
    m = km * 1000.0;
    cm = km * 100000.0;
    printf("Distance in Meter is %f\n", m);
    printf("Distance in Centimeter is %f\n", cm);
    return 0;
}
```

### OUTPUT

Enter distance in Kilometer 5

Distance in Meter is 5000.000000

Distance in Centimeter is 500000.000000

### 3. Program to check the given character is Lowercase or Uppercase or Special character

#### ALGORITHM

Algorithm given below to find out that a given character is upper case, lower case, number or special character.

Step1- Start

Step 2 – Read input character from console at runtime.

Step 3 – Compute ASCII value of the character.

Step 4 – If the ASCII value of the character is in the range of 65 and 90,

Then, print "Upper Case letter".

Step 5 – If the ASCII value of the character is in the range of 97 and 122,

Then, print "Lower Case letter".

Step 6 – If the ASCII value of the character is in the range of 48 and 57,

Then, print "Number".

Step 7 – Else, print "Symbol".

#### Source Code

```
#include<stdio.h>

int main( )
{
    char ch;
    printf("enter a character:");
    scanf("%c",&ch);
    if(ch >= 65 && ch <= 90)
        printf("Upper Case Letter");
    else if(ch >= 97 && ch <= 122)
        printf("Lower Case letter");
    else if(ch >= 48 && ch <= 57)
        printf("Number");
    else
        printf("Symbol");
    return 0;
}
```

#### OUTPUT

Enter a character:A  
Upper Case Letter

Enter a character:a  
Lower Case letter

Enter a character:@  
Symbol

Enter a character:1  
Number

### 3 Source code

```
#include<stdio.h>
int main( )
{
    char ch;

    /* Input character from user */
    printf("Enter any character: ");
    scanf("%c", &ch);
    /* Alphabet check */
    if(ch >= 'a' && ch <= 'z')
    {
        printf("%c' is lowercase.", ch);
    }
    else if(ch >= 'A' && ch <= 'Z')
    {
        printf("%c' is Uppercase.", ch);
    }
    else if(ch >= '0' && ch <= '9')
    {
        printf("%c' is digit.", ch);
    }
    else
    {
        printf("%c' is special character.", ch);
    }

    return 0;
}
```

### OUTPUT

Enter any character: A  
'A' is Uppercase.

Enter any character: a  
'a' is lowercase.

Enter any character: 2  
'2' is digit.

Enter any character: #  
'#' is special character.

**4. Program to balance the given Chemical Equation values x, y, p, q of a simple chemical equation of the type: The task is to find the values of constants b1, b2, b3 such that the equation is balanced on both sides and it must be the reduced form.**

```
#include<stdio>
int main()
{
int k,l,m,n;
printf("Enter the atomic nature of reactants x and y ");
scanf("%d %d", &k, &l);
printf("Enter the atomic nature of products p and q");
scanf("%d %d", &m, &n);
balance(k,l,m,n);
}
int balance(int x, int y, int p, int q)
{
int b1,b2,b3,temp;
if(p%x==0 && q%y==0)
{
b1=p/x;
b2=q/y;
b3=1;
}
else
{
p=p*y;
q=q*x;
b3=x*y;
temp=gcd(p,gcd(q,b3));
b1=p/temp;
b2=q/temp;
b3=b3/temp;}
printf("the coefficients are b1=%d, b2=%d, b3=%d",b1,b2,b3);
}
int gcd(int a, int b)
```

```
{  
  
int hcf;  
for( int i=1;i<=a && i<=b;i++)  
{  
if(a%i= =0 && b%i= =0)  
{  
hcf=i;  
}  
}  
  
return hcf;  
}
```

### **OUTPUT**

Enter the atomic nature of reactants x and y : 1 2

Enter the atomic nature of products p and q : 2 3

The coefficients are b1= 4, b2=3, b3=2

**5. Implement Matrix multiplication and validate the rules of multiplication. Algorithm: Matrix Multiplication**

This Algorithm computes and outputs the product of 2 matrices. The input matrices A is of the order  $M \times N$ , B is of the order  $P \times Q$  and the resultant matrix C is of the order  $M \times Q$ .  $M, N, P, Q$  are of type integers, the loop control variables I, J, K are of types.

Step 1 : [Begin]

Start

Step 2 : [Prompt]

Write("Enter the order of matrix A")

Step 3 : [Input order of matrix A]

Read (M,N)

Step 4 : [Prompt]

Write("Enter the order of matrix B")

Step 5 : [Input order of matrix B]

Read (P,Q)

Step 6 : [Check for matrix order compatibility]

If ( $N \neq P$ ) Then

Write("Invalid order")

Stop

End If

Step 7 : [Input elements to matrix A]

Repeat for  $i \leftarrow 0$  to  $M-1$ , Increment  $i$  in step of 1

Repeat for  $j \leftarrow 0$  to  $N-1$ , Increment  $j$  in step of 1

Read ( $A[i][j]$ )

End for

End for

Step 8 : [Input elements to matrix B]

Repeat for  $i \leftarrow 0$  to  $P-1$ , Increment  $i$  in step of 1

Repeat for  $j \leftarrow 0$  to  $Q-1$ , Increment  $j$  in step of 1

Read ( $B[i][j]$ )

End for

End for

Step 9 : [Process]

[Input elements to matrix A]

```
Repeat for i ← 0 to M-1, Increment i in step of 1
    Repeat for j ← 0 to Q-1, Increment j in step of 1
        C[i][j] ← 0
        Repeat for K ← 0 to N-1,
            Increment K in step of 1
            C [i][j] ← C[i][j]+(A[i][k] * B[k][j])
        End for
    End for
End for
```

Step 10 : [Output elements of matrix A]

```
Repeat for i ← 0 to M-1, Increment i in step of 1
    Repeat for j ← 0 to N-1, Increment j in step of 1
        Write (A[i][j])
    End for
    Write ('\n')
End for
```

Step 11 : [Output elements of matrix B]

```
Repeat for i ← 0 to P-1, Increment i in step of 1
    Repeat for j ← 0 to Q-1, Increment j in step of 1
        Write (B[i][j])
    End for
    Write ('\n')
End for
```

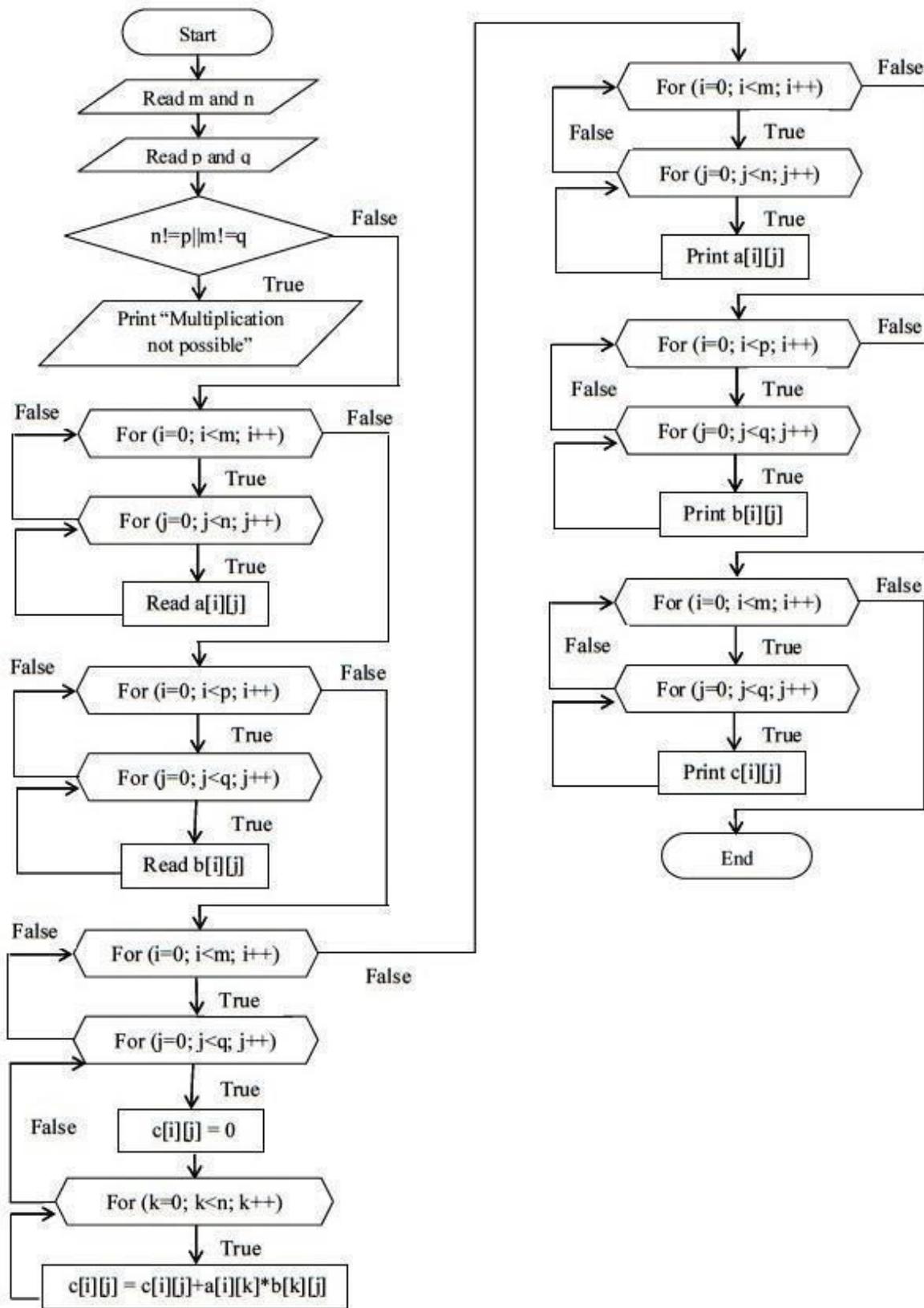
Step 12 : [Output elements of matrix C]

```
Repeat for i ← 0 to M-1, Increment I in step of 1
    Repeat for j ← 0 to Q-1, Increment J in step of 1
        Write (C[i][j])
    End for
    Write ('\n')
End for
```

Step 13: [Finished]

Stop

**Flowchart: Matrix Multiplication**



```
/*Program 5: Matrix Multiplication*/
#include<stdio.h>
int main()
{
int m,n;
int p,q;
printf("Enter the value of M and N\n");
//input order of matrix A
scanf("%d%d",&m,&n);
printf("Enter the value of P and Q\n");
//input order of matrix B
scanf("%d%d",&p,&q);
//declare, and define arrays a,b,c
int a[m][n];
int b[p][q];
int c[m][q];
//check the matrices order for compatibility
if(n!=p)
{
printf("Invalid order\n");
//finished
return 0;
}
printf("Enter %d elements into matrix A\n",(m*n));
//input elements
for(int i=0;i<m;i++)
for(int j=0;j<n;j++)
scanf("%d",&a[i][j]);
printf("Enter %d elements into matrix B\n",(p*q));
//input elements
for(int i=0;i<p;i++)
for(int j=0;j<q;j++)
```

```
scanf("%d",&b[i][j]);

//compute product
for(int i=0;i<m;i++)
{
for(int j=0;j<q;j++)
{
    c[i][j]=0;
for(int k=0;k<n;k++)
    c[i][j]+=a[i][k]*b[k][j]; //End of inner loop
}
}

//output matrix A
printf("Matrix A\n");
for(int i=0;i<m;i++)
{
for(int j=0;j<n;j++)
{
printf("%-4d",a[i][j]);
}
printf("\n");
}

//output matrix B
printf("Matrix B\n");
for(int i=0;i<p;i++)
{
for(int j=0;j<q;j++)
{
printf("%-4d",b[i][j]);
}
printf("\n");
}

//output resultant matrix C
printf("Matrix C\n");
for(int i=0;i<m;i++)
```

```
{  
for(int j=0;j<q;j++)  
{  
printf("%-4d",c[i][j]);  
}  
printf("\n");  
}  
return 0;  
}
```

### **Output**

Enter the value of M and N

2 2

Enter the value of P and Q

2 2

Enter 4 elements into matrix A

1 2 2 1

Enter 4 elements into matrix B

2 4 4 2

Matrix A

1 2

2 1

Matrix B

2 4

4 2

Matrix C

10 8

8 10

Enter the value of M and N

2 3

Enter the value of P and Q

2 2

Invalid order

**6. Compute  $\sin(x)/\cos(x)$  using Taylor series approximation. Compare your result with the built-in library function. Print both the results with appropriate inferences.**

**Algorithm:**

**TaylorSeries(sin(x))**

Step 1: Start

Step 2: [Initialize] Sin\_x= 0

Step 3: Read the value of x

Step 4: Convert x to radians ( $x*3.14/180$ )

Step 5: Compute sin(r)

for(j=1; j<=15; j=j+2)

{

neg = neg\*(-1);

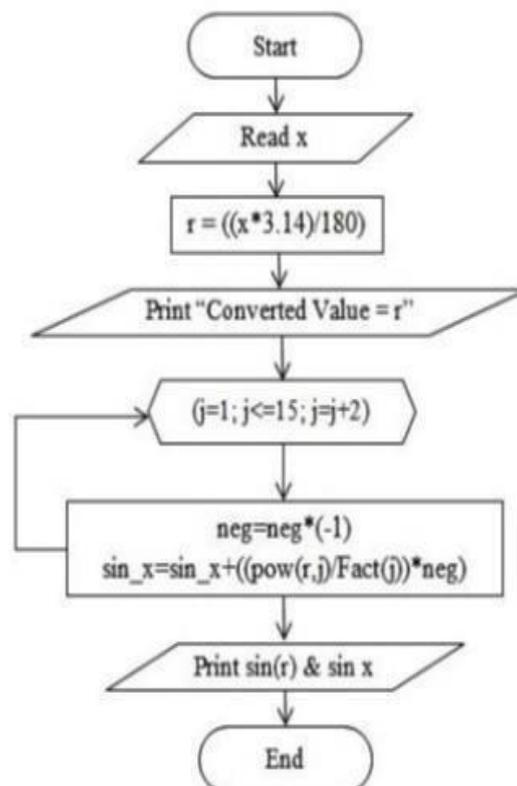
sin\_x = sin\_x + ((pow(r, j)/fact(j))\*neg);

print built-in value and calculated value

}

Step 6: Stop

**Flowchart:** Taylor Series



```
/*Program6:Taylor
Series(sin(x))*/
#include<stdio.h>
#include<math.h>
#define PI 3.142
int main( )
{
int i, degree;
float x, sum=0,term,nume,deno;
printf("Enter the value of degree");
scanf("%d",&degree);
x = degree * (PI/180); //converting degree into radian
nume= x;
deno= 1;
i=2;
do
{ //calculating the sine value.
term = nume/deno;
nume = -nume*x*x;
deno = deno*i*(i+1);
sum=sum+term;
i=i+2;
} while (fabs(term) >= 0.00001); // Accurate to 4 digits
printf("The sine of %d is %.3f \n", degree, sum);
printf("The sine function of %d is %.3f", degree, sin(x));
return 0;
}
```

### **Output**

Enter the value in

degree 90

sin(90)=1, by Taylor

series sin(90)=1, by

Built-in function

```
/*Program6:TaylorSeries(cos(x))*/  
#include<stdio.h>  
#include<math.h>  
#include<float.h>  
int  
main()  
{  
float  
deg=0,nterm,x,rvx;  
int i=1;  
printf("enter the value in  
degree\n"); scanf("%f",&x);  
rvx=x;  
  
x=x*(M_PI/  
180);  
nterm=1;  
while(fabs(nterm)>=FLT_EPSILON)  
{  
deg=deg+nterm;  
nterm=-nterm*(x*x)/((2*i-  
1)*(2*i)); i=i++;  
}  
}
```

```
printf("cos(%g)=%g, by Taylor series\n",rvx,deg);
printf("cos(%g)=%g, by Built-in
function\n",rvx,cos(x)); return 0;
}
```

### **OUTPUT**

Enter the value in degree 60

cos(60)=0.5, by Taylor series

cos(60)=0.5, by Built-in function

**7. Sort the given set of N numbers using Bubble sort.**

**Algorithm:**

**Step 1 :** Start

**Step 2 :** [ Enter a number]Read n

**Step 3:** [ Enter the elements ]for i=0 to n

    read a[i]

**Step 4:** [print the elements ]

    for i=0 to n

        print a[i]

**Step 5:** [sort the elements ]

/\* terminate the inner loop first then update the outer loop]

    for i=1 to n [number of passes required]

        for j=0 to n-i [access the element in each pass]

            if(a[j]>a[j+1]) then

end for

end if

        temp=a[j];

        a[j]=a[j+1];

        a[j+1]=temp;

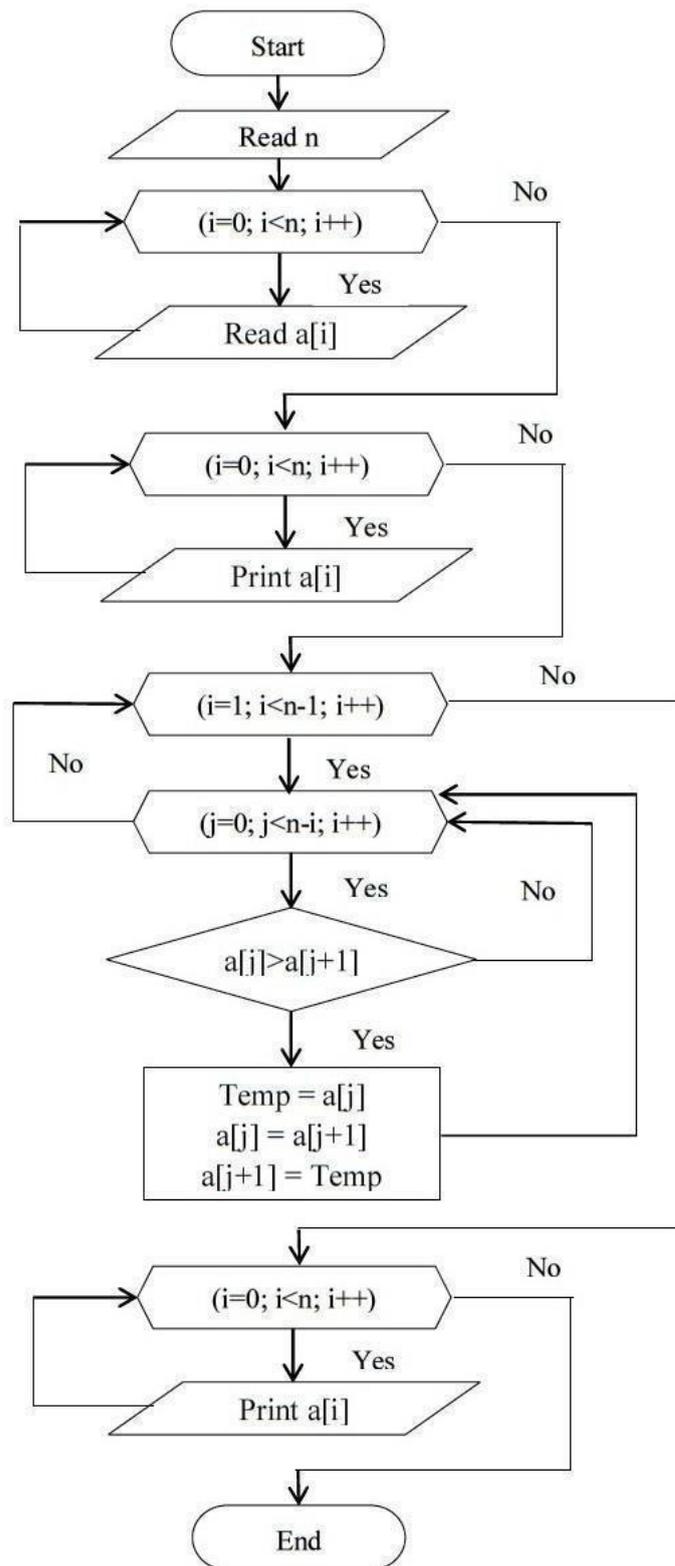
end for

**Step 6:** [ print the sorted elements ]

    for i=0 to n

        print a[i]

**Step 7 :** Stop



**Program**

```
#include <stdio.h>
#include <stdlib.h>
void main( )
{
int a[50],i,j,temp,n; /* Declaration of the variables*/
printf("Enter the value of n\n");
scanf("%d",&n); /* Read the size of an array */
printf("Enter the array elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]); /* Read the elements */
printf("The given array elements are \n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
/* Perform Bubble Sort */
for(i=1;i<n;i++)
{
for(j=0;j<n-i;j++)
{
if(a[j]>a[j+1])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}
}
printf("\nThe array after sorting\n"); /* Display the array after sorting */
for(i=0;i<n;i++)
printf("%d\t",a[i]);
}
```

**OUTPUT**

```
Enter the value of n : 5
Enter the array elements
15 10 5 25 20
The given array elements are
15 10 5 25 20
The array after sorting
5 10 15 20 25
```

**8. Write functions to implement string operations such as compare, concatenate, stringlength. Convince the parameter passing techniques.**

**Algorithm**

Step 1: Start

Step 2: Read strings s1 and s2

Step 3: Call function string\_length(), length1=string\_length(s1)

    Lenght2= string\_length(s2)

Step 4: Display length1 and length2

Step 5: Call function compare\_string()

    If(compare\_string(s1,s2)==0)

        Print "Equal Strings"

Else

    Print "Unequal Strings"

Step 6: Call function concatenate(s1,s2)

Step 7: Print "Concatnated String"

Step 8: Stop

**Function : string\_lenth(char s[])**

Step 1: Start

Step 2: Repeat step2 through while(s1[c]!='\0')

    C++

    Return c

    End while

Step 3: Stop

Function: compare\_strings(char s1[], char s2[])

Step 1: start

Step2: Reapeat step2 through while (s1[c]==s2[c])

If(s1[c]=='\0' || s2[c]=='\0')

Break;

C++

End while

Step 3: if( s1[c]=='\0' && s2[c]=='\0')

return 0

Else

return 1

Step 4: Stop

Function:Concatenate(char s1[],char s2[])

Step 1: Start

Step 2: Initialize c=0

Step 3: Repeat steps through while(s1[c]!='\0')

c++

end while

Step 4: Initialize d=0

Step 5: Repeat step5 through while (s2[d]!='\0')

S1[c]=s2[d]

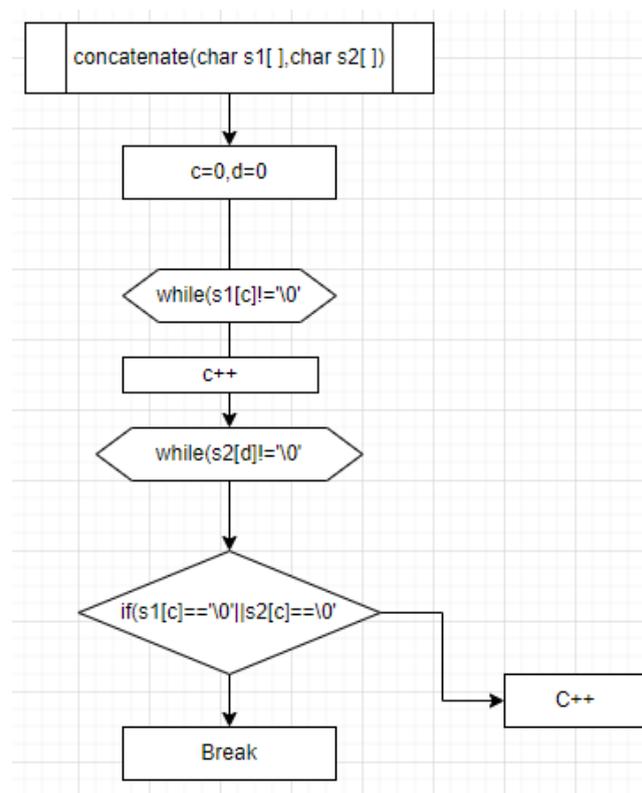
d++

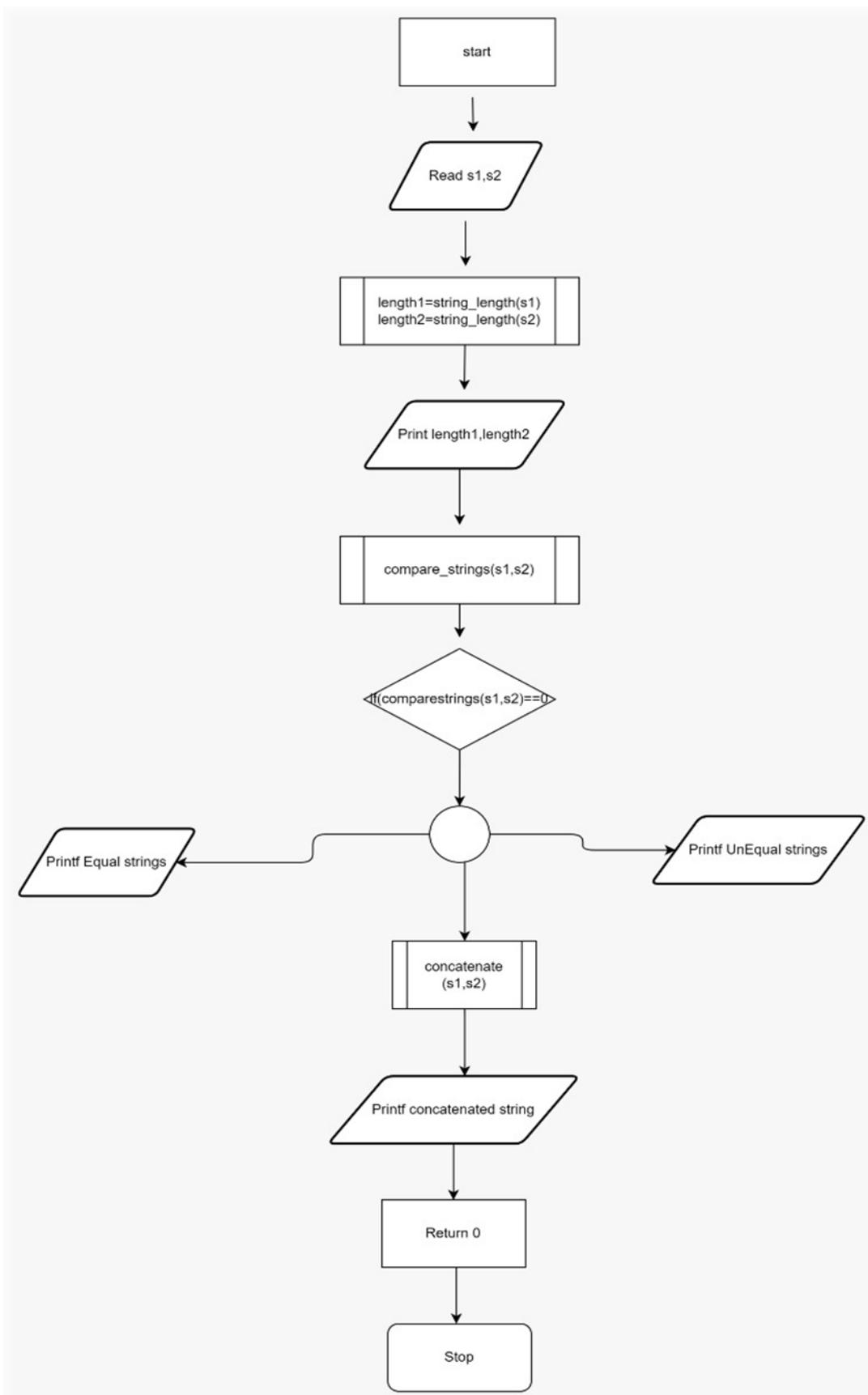
c++

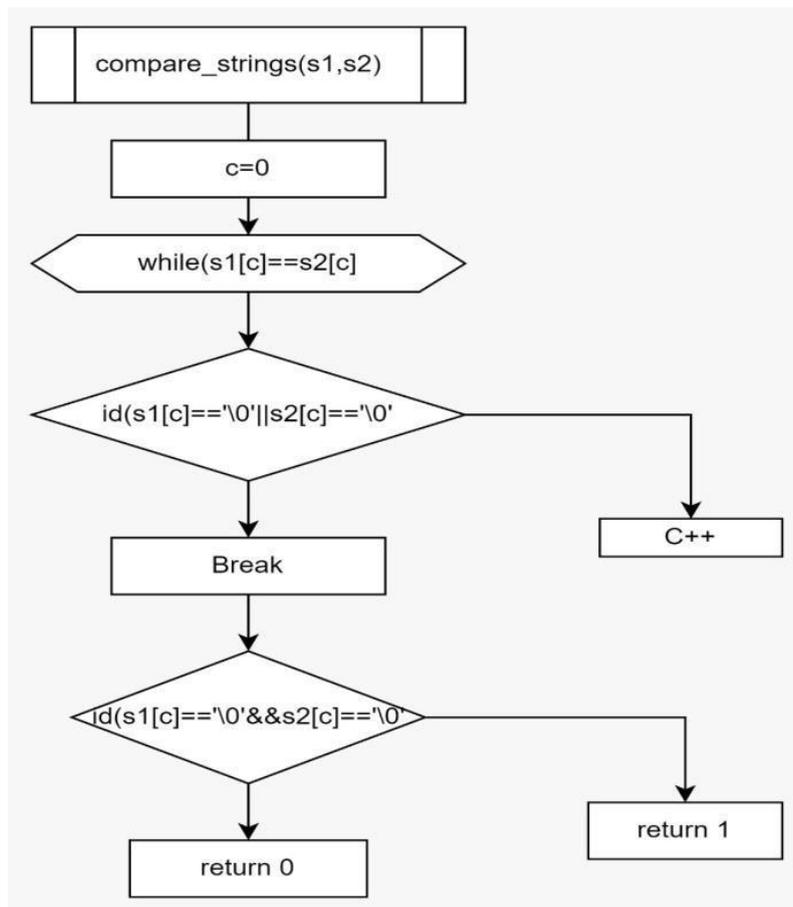
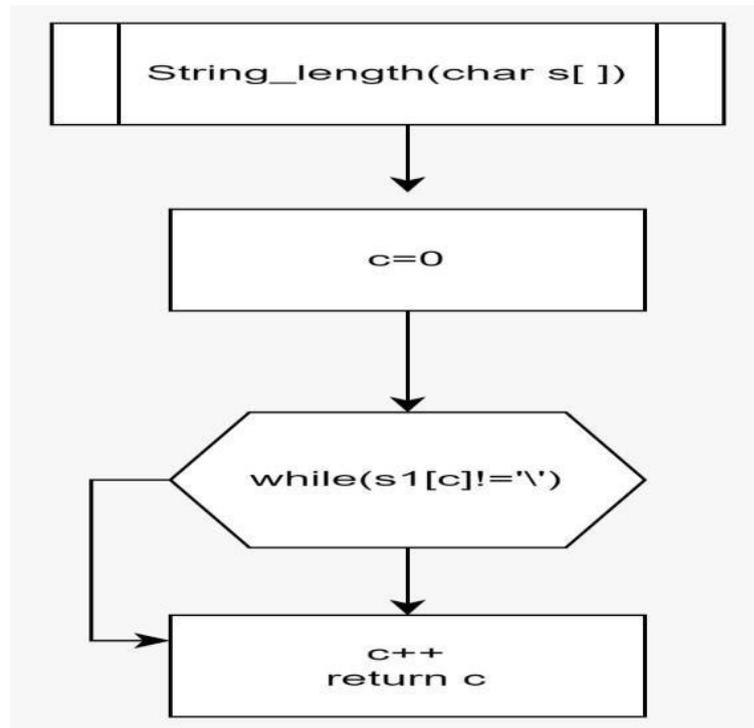
end while

step 6: s1[c]='\0'

Step 7: Stop







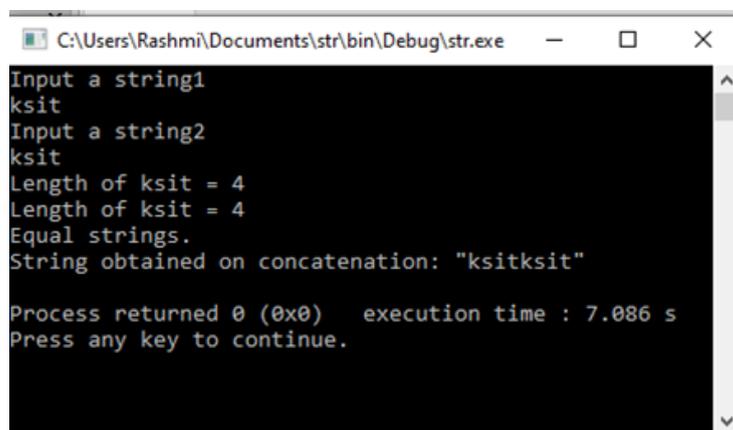
```
/*Program*/
#include <stdio.h>
#include <stdlib.h>
int compare_strings(char [], char []);
void concatenate(char [], char []);
int main()
{
    char s1[1000],s2[1000];
    printf("Input a string1\n");
    gets(s1);
    printf("Input a string2\n");
    gets(s2);
    int length1 = string_length(s1);
    int length2 = string_length(s2);
    printf("Length of %s = %d\n", s1,length1);
    printf("Length of %s = %d\n", s2,length2);
    if (compare_strings(s1, s2) == 0)
        printf("Equal strings.\n");
    else
        printf("Unequal strings.\n");
    concatenate(s1,s2);
    printf("String obtained on concatenation: \"%s\"\n", s1);
    return 0;
}

/* string length function*/
int string_length(char s1[])
{
    int c = 0;
    while (s1[c] != '\0')
        c++;
    return c;
}
```

```
int compare_strings(char s1[], char s2[])
{
    int c = 0;
    while (s1[c] == s2[c])
    {
        if (s1[c] == '\0' || s2[c] == '\0')
            break;
        c++;
    }
    if (s1[c] == '\0' && s2[c] == '\0')
        return 0;
    else
        return 1;
}
```

```
void concatenate(char s1[], char s2[])
{
    int c, d;
    c = 0;
    while (s1[c] != '\0')
    {
        c++;
    }
    d=0;
    while(s2[d]!='\0')
    {
        s1[c] = s2[d];
        d++;
        c++;
    }
    s1[c] = '\0';
}
```

### OUTPUT



**9. Implement structures to read, write and compute average marks and the students scoring above and below the average marks for a class of N students.**

**Algorithm**

Step 1: Start

Step 2: Create a structure with student including fields usn,name and marks.

Step 3: Initialize the variables countav=0,countbv=0

Step 4: Read the number of students n

Step 5: Read the value of usn, name and marks for the specified no of students using structure variable s[i] for i=0 to n-1

Step 6: Display the details of students

Step 7: Repeat for i=0 to n-1

    Compute sum of the marks

Step 8: Compute the average

Step 9: Repeat for i=0 to n-1

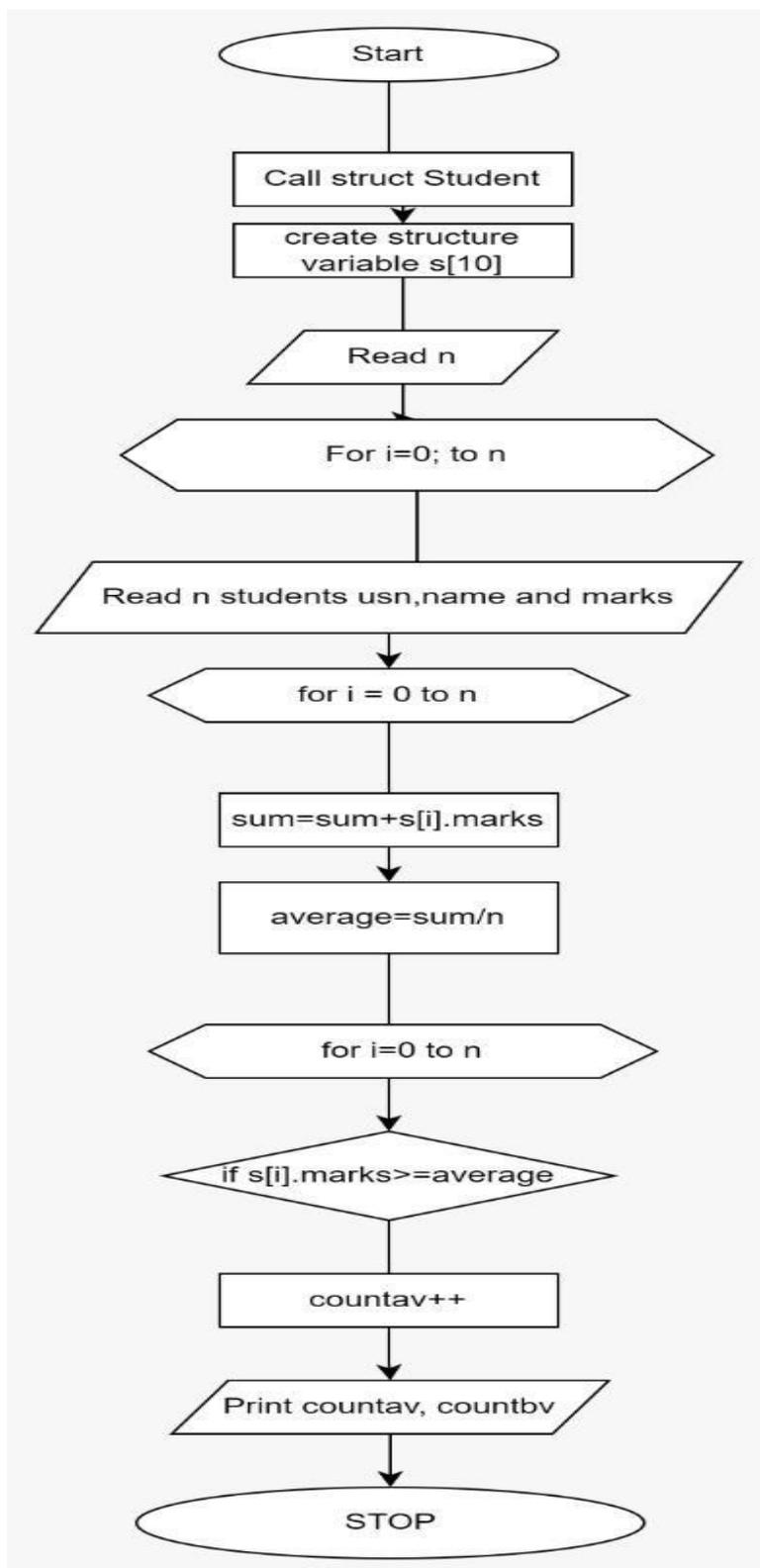
    If(s[i].marks>=average)

        Print " Total no of students above average"

    Else

        Print " Total number of students below average"

Step 10: Stop



**/\*PROGRAM**

```
#include <stdio.h>
#include <stdlib.h>

struct student
{
char usn[50];
char name[50];
int marks;
} s[10];
void main()
{
int i,n,countav=0,countbv=0;
float sum,average;
printf("Enter number of Students\n");
scanf("%d",&n);
printf("Enter information of students:\n");
// storing information
for(i=0; i<n;i++)
{
printf("Enter USN: ");
scanf("%s",s[i].usn);
printf("Enter name: ");
scanf("%s",s[i].name);
printf("Enter marks: ");
scanf("%d",&s[i].marks);
printf("\n");
}
printf("Displaying Information:\n\n");
// displaying information
for(i=0; i<n; i++)
{
printf("\nUSN: %s\n",s[i].usn);
printf("Name: %s\n ", s[i].name);
printf("Marks: %d",s[i].marks);
printf("\n");
}
for(i=0;i<n;i++)
{
sum=sum+s[i].marks;
}
average=sum/n;
printf("\nAverage marks: %f",average);
for(i=0;i<n;i++)
{
if(s[i].marks>=average)
countav++;
else
countbv++;
}
```

```
}  
printf("\nTotal No of students above average= %d",countav);  
printf("\nTotal No of students below average= %d",countbv);  
}
```

## OUTPUT

```
Enter number of Students  
2  
Enter information of students:  
Enter USN: 1  
Enter name: aaa  
Enter marks: 90  
  
Enter USN: 2  
Enter name: bbb  
Enter marks: 60  
  
Displaying Information:  
  
USN: 1  
Name: aaa  
Marks: 90  
  
USN: 2  
Name: bbb  
Marks: 60  
  
Average marks: 75.000000  
Total No of students above average= 1  
Total No of students below average= 1  
Process returned 38 (0x26)   execution time : 15.831 s  
Press any key to continue.  
█
```

**10. Develop a program using pointers to compute the sum, mean and standard deviation of all elements stored in an array of N real numbers.**

Algorithm

Step1:Start

Step 2: Read n

Step 3: Repeat for i=0 to n-1

    Read a[i]

Step 4: ptr=a

Step 5: Repeat for i=0 to n-1

    Sum=sum+\*ptr

    ptr++

Step 6: mean=sum/n

Step 7: ptr=a

Step 8: Repeat for i=0 to n-1

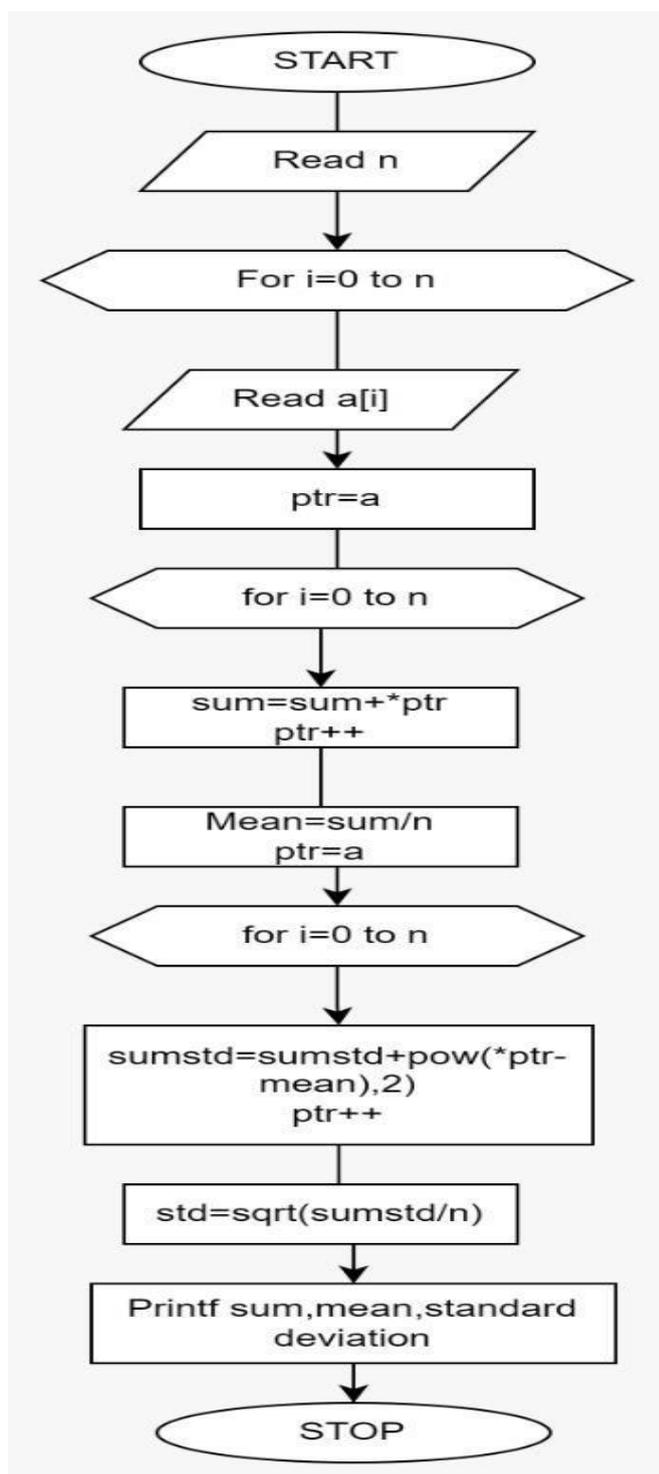
    Sumstd=sumstd+pow((\*ptr-mean),2)

    Ptr++

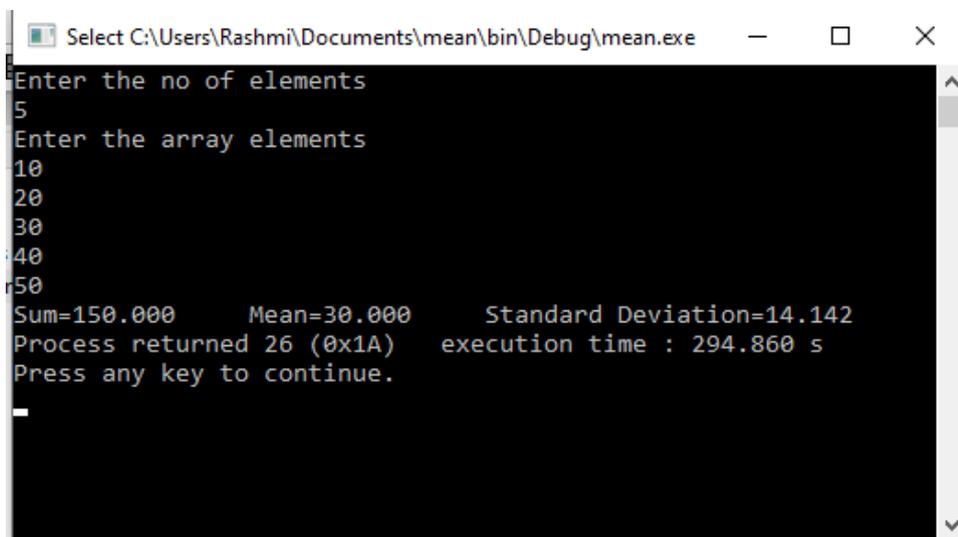
Step 9: std=sqrt(sumsrd/n)

Step 10: Print sum,mean and standard

deviationStep 11:Stop



```
/*Program*/
#include <stdio.h>
#include <stdlib.h>
#include<math.h>
void main()
{
float a[10],*ptr,mean,std,sum=0,sumstd=0;
int n,i;
printf("Enter the no of elements\n");
scanf("%d",&n);
printf("Enter the array elements\n");
for(i=0;i<n;i++)
{
scanf("%f",&a[i]);
}
ptr=a;
for(i=0;i<n;i++)
{
sum=sum+*ptr;
ptr++;
}
mean=sum/n;
ptr=a;
for(i=0;i<n;i++)
{
sumstd=sumstd+pow((*ptr-mean),2);
ptr++;
}
std=sqrt(sumstd/n);
printf("Sum=%.3f\t",sum);
printf("Mean=%.3f\t",mean);
printf("Standard Deviation=%.3f\t",std);
}
```



```
Select C:\Users\Rashmi\Documents\mean\bin\Debug\mean.exe
Enter the no of elements
5
Enter the array elements
10
20
30
40
50
Sum=150.000 Mean=30.000 Standard Deviation=14.142
Process returned 26 (0x1A) execution time : 294.860 s
Press any key to continue.
```

