## PROGRAM-1

**Simulation of a Simple Calculator.**

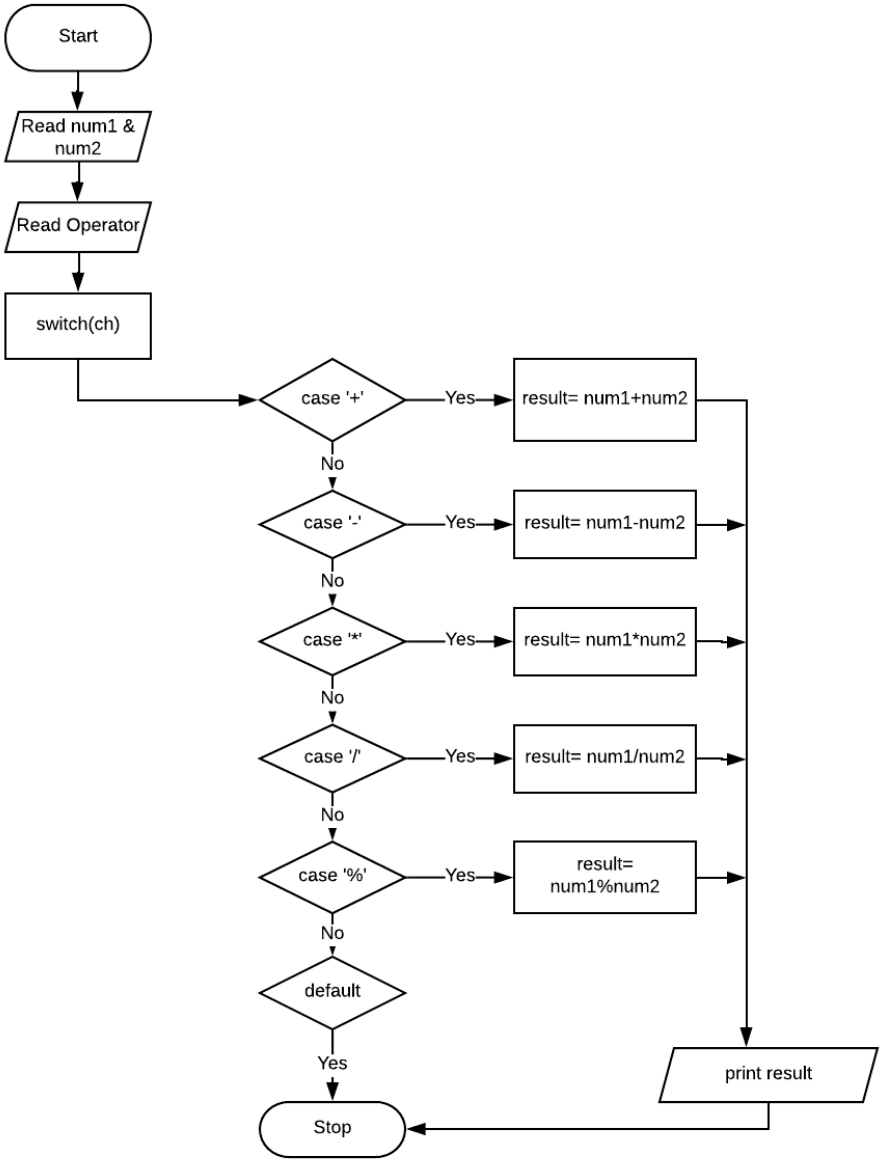
**Procedure**: This program takes an arithmetic operator +,-,*,/,% and two operands from the user and performs the calculation on the two operands depending upon the operator entered by the user.
**Input:** An operator and two operands.
**Expected Output:** Performs calculation and display result depending upon the operator.

## ALGORITHM

Algorithm Calculator

Step 1: Start
Step 2: Read num1 and num2
Step 3: Enter the operator
Step 4: Evaluate operator wit case statements
    Step 4.1: case '+' : result = num1+num2
        goto step 6
        Print result
    Step 4.2: case '-' : result = num1-num2
        goto step 6
        Print result
    Step 4.3: case '*' : result = num1*num2
        goto step 6
        Print result
    Step 4.4: case '/' : result = (float)num1/(float)num2
        goto step 6
        Print result
    Step 4.5: case '%' : result = num1%num2
        goto step 6
        Print result
Step 5: Enter operator is invalid then
    Print "Invalid Operation"
Step 6: Print result
Step 7: Stop

## FLOWCHART




## PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{
   int num1, num2;
   float result=0;
   char ch;
   printf("Choose operation to perform (+,-,*,/,%): ");
   scanf(" %c", &ch);

   printf("Enter first number: ");
```

```
    scanf("%d", &num1);
    printf("Enter second number: ");
    scanf("%d", &num2);

    switch(ch)
    {
       case '+':result=num1+num2;
          break;
       case '-':result=num1-num2;
          break;
       case '*': result=num1*num2;
          break;
       case '/': if(num2==0)
                    {
                          printf("Error: divid by zero\n");
                          exit(0);
                    }
                    result=(float)num1/(float)num2;
                     break;
      case '%': result=num1%num2;
          break;
     default: printf("Invalid operation.\n");
               exit(0);
    }
  printf("Result: %d %c %d = %f\n",num1,ch,num2,result);     //display output on screen

}
```

**INPUT-OUTPUT:**

Choose operation to perform (+,-,*,/,%): +
Enter first number: 10
Enter second number: 25
Result: 10 + 25 = 35.000000

Choose operation to perform (+,-,*,/,%): /
Enter first number: 12
Enter second number: 0
Error: divid by zero

## PROGRAM-2

**Compute the roots of a quadratic equation by accepting the coefficients. Print appropriate messages.**

**Procedure**: The equation in the form $ax^2+bx+c=0$ is called quadratic equation. Read the coefficients a,b,c and calculate discriminant. Based on the discriminant value, calculate roots and print them with suitable messages.

**Input**: Three coefficients of quadratic equation $ax^2+bx+c=0$: a, b, c

**Expected Output**: This program computes all possible roots for a given set of coefficients with appropriate messages. The possible roots are: Real and Equal roots, Real and distinct roots, imaginary roots.

## ALGORITHM

Algorithm: Quadratic_Equation [This algorithm takes three coefficients as input and compute the roots]

Step 1: [Start of the algorithm]

Start

Step 2: [Read the coefficients]

Read non zero coefficients a,b,c

Step 3: [calculate the discriminant]

d←b*b-4*a*c

Step 4: [check if roots are real and equal]

if (d=0)

x1←-b/(2*a)

x2←-b/ (2*a)

Print “Roots are equal”

Print x1, x2

Go to step 7

Step 5: [check if roots are real and distinct]

If(d>0)

x1←(-b+sqrt (d)/ (2*a))

x2←(-b-sqrt (d)/ (2*a))

Print “Roots are real and distinct”

Print x1,x2

Go to step 7

Step 6: [check if roots are imaginary]

If(d<0)

x1←-b/(2*a)

x2←sqrt (fabs(d))/(2*a)

Print " The roots are complex"

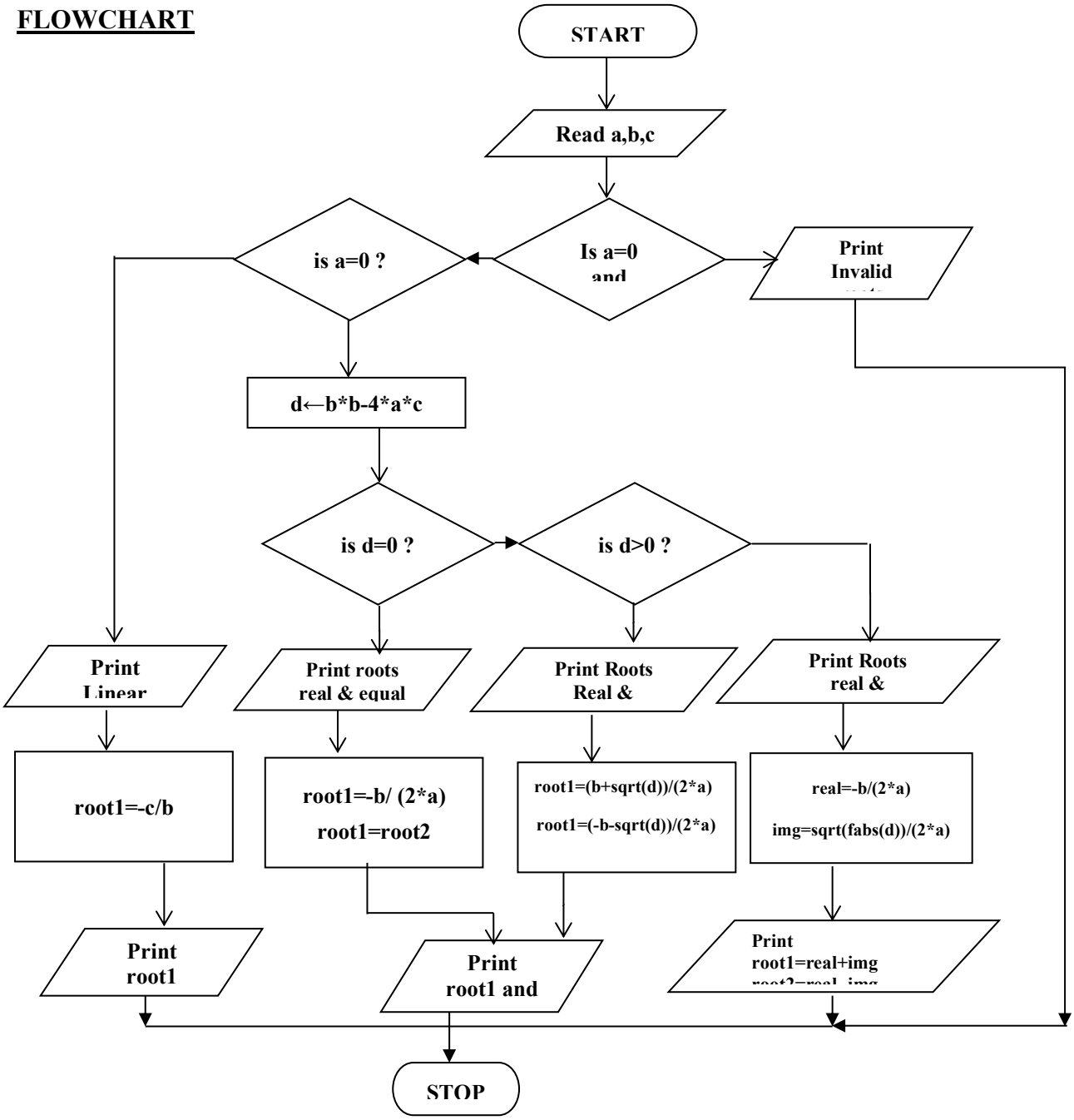
Print "Root1 ← ", x1+ix2

Print " Root2←", x1-ix2

Step 7: [terminate the algorithm]

Stop

**FLOWCHART**



**PROGRAM:**

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

int main()
{
	float a,b,c,d,rpart,ipart,root1,root2;
	printf("Enter three  co-efficient\n");
	scanf("%f%f%f",&a,&b,&c);

	if(a==0 && b==0)
	{
		printf(“Invalid inputs”);

	}
	else if(a==0)
	{
		printf(“Linear Equation\n”);
		root1=-c\b;
		printf(“Root=%f\n”,root1);
	}
	else
	{
		d=(b*b)-(4*a*c);
		if(d==0)
		{
			printf("The roots real and equal\n");
			root1= -b/(2*a);
			root2=root1;
			printf("The roots are root1=%.3f and root2=%.3f\n",root1,root2);
		}
		else if(d>0)
		{
			printf("The roots are real and distinct\n");
			root1=(-b+sqrt(d))/(2*a);
			root2=(-b-sqrt(d))/(2*a);
			printf("The roots are root1=%.3f and root2=%.3f\n",root1,root2);
		}
		else
		{
			printf("The roots are imaginary\n");
			rpart=-b/(2*a);
			ipart=sqrt(fabs(d))/(2*a);
			printf("The first root root1=%.3f+i%.3f\n",rpart,ipart);
			printf("The second root root2=%.3f-i%.3f\n",rpart,ipart);
		}
	}
}
```

**OUTPUT:**

.....

Run1:

Enter three co-efficient
1
2
3
The roots are imaginary
The first root root1=-1.000+i1.414
The second root root2=-1.000-i1.414

Run2:

Enter three co-efficients
1
5
2
The roots are real and distinct
The roots are root1=-0.438 and root2=-4.561

Run3:

Enter three co-efficients
1
2
1
The roots real and equal
The roots are root1=-1.000 and root2=-1.000

## PROGRAM-3

**An electricity board charges the following rates for the use of electricity: for the first 200 units 80 paise per unit: for the next 100 units 90 paise per unit: beyond 300 units Rs 1 per unit. All users are charged a minimum of Rs. 100 as meter charge. If the total amount is more than Rs 400, then an additional surcharge of 15% of the total amount is charged. Write a program to read the name of the user, the number of units consumed, and print out the charges.**

**Input:** Consumer name and number of units

**Expected Output:** This program checks all else if ladder conditions as per number of units consumed by consumer and display appropriate output

## ALGORITHM

Algorithm power

Step 1: Start

Step 2: Read name & unit

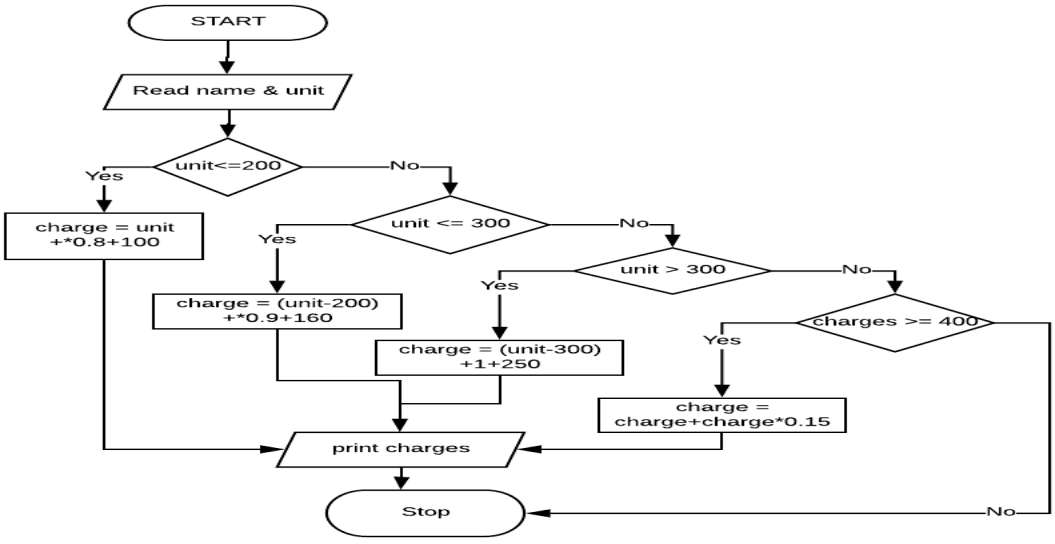
Step 3: if(unit <=200)

```
        charge = unit *0.8+ 100
        else if(unit<=300)
                charge=(unit-200) *0.90+160;
        else if(unit>300)
                charge=(unit-300) *1+250;
        if(charge>=400)
                charge=charge + charge*0.15;
```

Step 4: Display the charges

Step 5: Stop

## FLOWCHART



**PROGRAM:**

```
#include <stdio.h>
int main()
{
char name[10];
float unit, charge, min=100 ;
printf("Enter your name and unit Consumed:\n");
scanf("%s%f",name, &unit);

 if(unit<=200)
       charge=unit*.80+min;
else if(unit<=300)
       charge=(unit-200)*0.90+160+min;
else
{
if(unit>300)
       charge=(unit-300) *1+250+min;
}
if(charge>=400)
       charge=charge + charge*0.15;

printf("Name: %s\n Charge: %.3f\n", name, charge);

}
```

**OUTPUT:**

Enter your name and unit Consumed:

Arun

340

Name: Arun

Charge:390

## PROGRAM-4

**Write a C Program to display the following by reading the number of rows as input,**

**1**
**1 2 1**
**1 2 3 2 1**
**1 2 3 4 3 2 1**
**----------**
**$n^{th}$ row**

**PROGRAM:**

```
#include<stdio.h>
void main()
{
int row,col,sp,ans,no;
printf("enter no. of rows\n");
scanf("%d",&no);
for(row=0;row<no;row++)
{
      for(sp=0;sp<(no-row);sp++)
      {
            printf("  ");
      }
      ans=1;
      for(col=0;col<=row;col++)
      {
            printf("  %d",ans);
            ans=ans*(row-col)/(col+1);
      }
      printf("\n");
}
}
```

**OUTPUT:**

```
Enter no. of rows
10
                1
              1  1
            1  2  1
          1  3  3  1
        1  4  6  4  1
      1  5  10  10  5  1
    1  6  15  20  15  6  1
  1  7  21  35  35  21  7  1
1  8  28  56  70  56  28  8  1
1  9  36  84  126  126  84  36  9  1
```

## PROGRAM-5

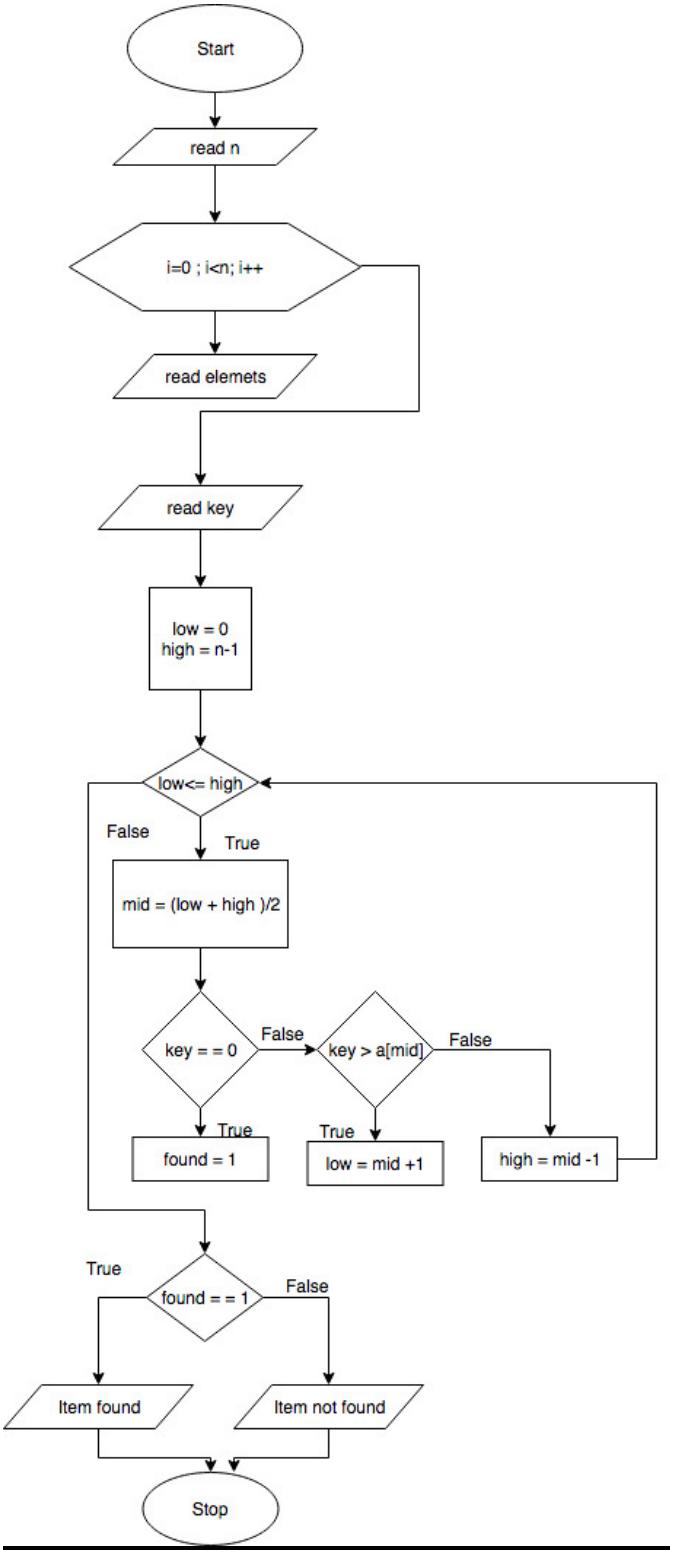
**Implement Binary Search on Integers .**

**Procedure:** Input N array elements and to find whether element is present or not.
**Input:** Array elements.
**Expected Output:** Successful search or unsuccessful search.

## ALOGRITHM

Algorithm bubble_sort
Step 1: Start
Step 2: Read n
Step 3: Repeat for i=0 to n-1
Read a[i]
Step 4: read key
Step 5: Initialize low =0 high = n-1
Step 6: Repeat through step 6 while (low <= high)
mid = (low+ high)/2
if(key==a[mid])
found=1
else if(key>a[mid])
low=mid+1
else
high=mid-1
end while
Step 7: if(found ==1)
print “Item found”
else
print” Item not found”
Step 8: Stop

## FLOWCHART


Start
read n
i=0 ; i<n; i++
read elemets
read key
low = 0
high = n-1
low<= high
False
True
mid = (low + high )/2
key = = 0
False
key > a[mid]
False
True
found = 1
True
low = mid +1
high = mid -1
True
found = = 1
False
Item found
Item not found
Stop

**PROGRAM:**

```
#include<stdio.h>
int main()
 {
  int i,n,a[10],mid,low,high,key, found=0;

  printf("\n Enter the number of elements:\n");
  scanf("%d", &n);
  printf("Enter the array element in the ascending order\n");
  for(i=0;i<n;i++)
   {
    scanf("%d", &a[i]);
   }

 printf("\n Enter the key element to be searched\n");
  scanf("%d", &key);

   low=0;
  high=n-1;
  while(low<=high)
   {
          mid=(low+high)/2;
         if(key==a[mid])
         {
                  found=1;
                  break;
         }
         else if(key>a[mid])
                  low=mid+1;
         else
                  high=mid-1;
}
 if(found ==1)
         printf("Item found in position : %d",mid+1);
else
         printf("\n Item not found\n");
}
```

**OUTPUT:**

**RUN 1:**

```
Enter the number of elements:
5
Enter the array element in the ascending order
10
20
30
40
50
Enter the key element to be searched
30
```

Item found in position: 3

**RUN 2:**

```
Enter the number of elements:
5
Enter the array element in the ascending order
10
20
30
40
50
Enter the key element to be searched
70
Item not found
```

## PROGRAM-6

**Implement Matrix multiplication and validate the rules of multiplication.**

**Procedure:** Input m*n and p*q size of 2 matrices elements to compute matrix multiplication.

## ALGORITHM

Algorithm: Matrix_Multipication

Step 1: [Start of the algorithm]
Start
Step 2: [Read the order of the matrix A]
Read m,n
Step 3: [Read the order of the matrix B]
Read p,q
Step 4: [To check for compatibility condition of matrix multiplication]
if(n!=p)
print "Matrix multiplication not possible"
go to step 11
Step 5: [Read the elements of matrix A]
Repeat through step 5 for i←0 to m-1
Repeat for j←0 to n-1
Read a[i][j]
Step 6: [Read the elements of matrix B]
Repeat through step 6 for j←0 to q-1
Repeat for i←0 to p-1
Read b[i][j]
Step 7: [Display the elements of matrix A]
Repeat through step 7 for i←0 to p-1
Repeat for j←0 to q-1
Print a[i][j]
Step 8: [Display the elements of matrix B]
Repeat through step 8 for i←0 to p-1
Repeat for j←0 to q-1
Print b[i][j]
Step 9: [Calculate the product of two given matrices]
Repeat through step 9 for i←0 to m-1
Repeat for j←0 to q-1
c[i][j]←0
Repeat for k←0 to n-1
c[i][j]←c[i][j]+a[i][k]*b[k][j]
Step 10: [Print the resultant matrix]
Repeat step 10 for i←0 to m-1
Repeat for j←0 to n-1
Print c[i][j]
Step 11: [terminate the algorithm]
Stop

**Program:**

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

```
#include<conio.h>
int main()
{
	int m,n,p,q,i,j,k,a[10][10],b[10][10],c[10][10];
	printf("Enter the size matrix A \n");
	scanf("%d%d",&m,&n);
	printf("Enter the size matrix B \n");
	scanf("%d%d",&p,&q);

	if(n!=p)
	{
		printf(“Matrix multiplication is not possible\n”);
	}
	else
	{
		printf("Enter the elements of matrix A \n");
			for(i=0;i<m;i++)
				{
				for(j=0;j<n;j++)
					scanf ("%d",&a[i][j]);
				}
			printf("Enter the elements of matrix B \n");
			for(i=0;i<p;i++)
				{
	`			for(j=0;j<q;j++)
					scanf("%d",&b[i][j]);
				}
			for(i=0;i<m;i++)
			{
				for(j=0;j<q;j++)
				{
					c[i][j]=0;
					for(k=0;k<n;k++)

					c[i][j]=c[i][j]+a[i][k]*b[k][j];
				}

			}
			printf(‘A-matrix is\n");
				for(i=0;i<m;i++)
				{
					for(j=0;j<n;j++)
					{
						printf("%d\t",a[i][j]);
					}
					printf("\n");
				}

				printf("B- matrix  is \n");
				for(i=0;i<p;i++)
				{
```

```
                                        for(j=0;j<q;j++)
                                        {
                                                printf("%d\t",b[i][j]);
                                        }
                                        printf("\n");
                                }

                                printf("The resultant matrix C is \n");
                                for(i=0;i<m;i++)
                                {
                                        for(j=0;j<q;j++)
                                        {
                                                printf("%d\t",c[i][j]);
                                        }
                                        printf("\n");
                                }
        }

}
```

**OUTPUT (PASS 1)**

**.....**

```
Enter the size of matrix A
2 3
Enter the size of matrix B
4 5
Matrix multiplication is not possible
```

**OUTPUT (PASS 2)**

```
Enter the size of matrix A
2 2
Enter the size of matrix B
2 2
Enter the elements of Matrix A
1       1
1       1
Enter the elements of Matrix B
1        1
1        1

Matrix-A is
        1       1
        1       1
Matrix-B is
        1       1
        1       1
The resultant matrix c is

        2       2
        2       2
```

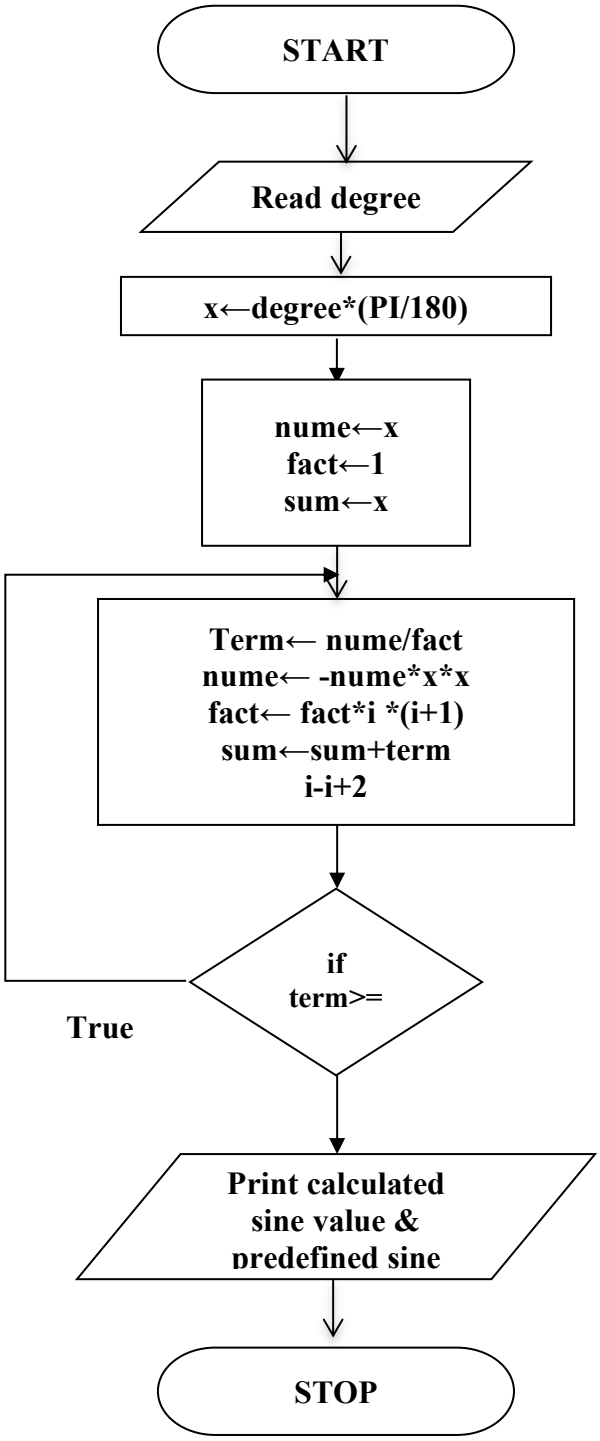
## PROGRAM-7

**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

Algorithm: Sine_series
Step 1:[Start the algorithm]
Start
Step 2:[Read the value of x]
Read x
Step 3:[Initialize the variables]
x1←x;
x←x*(3.142/180.0)
Step 4;[Read the value of no of terms]
Read n
Step 5:[Initialize the variables]
term←x;
sinx←term;
Step 6:[Repeat the following steps for n=1 to n terms
den←2*n*(2*n+1)
term←-term*x*x/den;
sum=sum+term;
Step 7:Print the result
sum of sine series=sum
sum using library function=sin(x)
Step 8: Stop

**FLOWCHART**


START
Read degree
x←degree*(PI/180)
nume←x
fact←1
sum←x
Term← nume/fact
nume← -nume*x*x
fact← fact*i *(i+1)
sum←sum+term
i-i+2
if
term>=
True
Print calculated
sine value &
predefined sine
STOP

**Program:**

```
#include<stdio.h>
#include<math.h>
#define PI 3.142

int main()
{
        int i, degree;
        float x,sum=0,term,nume,fact;
        printf("Enter value of degree \n");
        scanf("%d", &degree);

        x=degree*(PI/180);
        nume=x;
        fact=1;
        i=2;
        do
        {
                term= nume/fact;
                nume=-nume*x*x;
                fact=fact*i*(i+1);
                sum= sum+term;
                i=i+2;
        }while(fabs(term)>=0.00001);
        printf("The sine of %d is %.3f",degree,sum);
        printf("Using inbuilt function sin(%d)  is %.3f",degree,sin(x));

}
```

**OUTPUT:**

**.....**

```
Enter value of degree
0
The sine of 0 is 0.000
Using inbuilt function sin (0) is 0

Enter value of degree
60
The sine of 60 is 0.866
Using inbuilt function sin(60)  is 0.866

Enter value of degree
-10
The sine of -10 is -0.17
Using inbuilt function sin(-10)  is -0.17
```

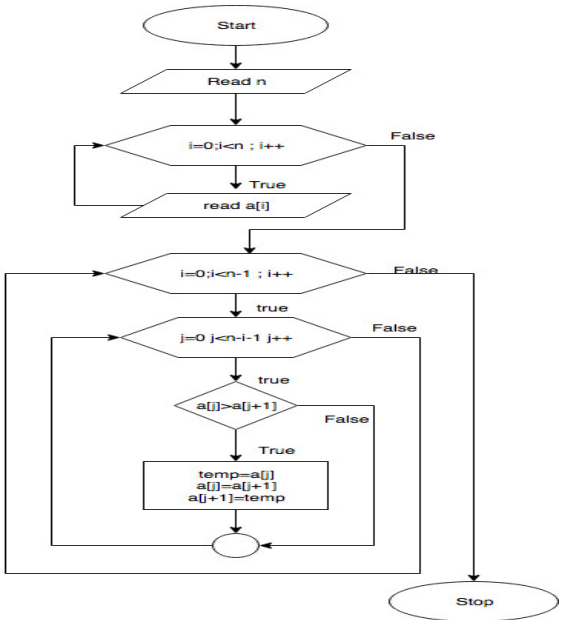
# PROGRAM-8

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

## ALGORITHM

Algorithm: Bubble sort
Step 1: [Start of the algorithm]
Start
Step 2: [Read the size of the array]
read n
Step 3: [Read the array elements]
Repeat for i=0 to n-1
read a[i]
Step 4: [Print the given array]
Repeat for i=0 to n-1
print a[i]
Step 5: Repeat through step 5 for i←1 to n-1
Repeat for j←0 to n-i
if(a[j]>a[j+1])
temp←a[j]
a[j] ←a[j+1]
a[j+1]←temp
Step 5: [Print the sorted array]
Repeat for i ←0 to n-1
print a[i]
Step 6 :[terminate the algorithm]
Stop

## FLOWCHART



**Program:**

```
#include<stdio.h>
int main()
{
        int a[100],n,i,j,temp;

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

        printf("Enter the %d elements of array\n",n);
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        printf("The Input array is\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t",a[i]);
        }
        for(i=0;i<n-1;i++)
        {               for(j=0;j<n-i-1;j++)
                        {       if(a[j]>a[j+1])
                                {
                                        temp=a[j];
                                        a[j]=a[j+1];
                                        a[j+1]=temp;
                                }
                        }

        }

        printf("\nThe sorted array is\n");
        for(i=0;i<n;i++)
        printf("%d\t",a[i]);

}
```

**OUTPUT:**

**.....**

Run1:

```
Enter the number of elements
4
Enter 4 elements of array
87
100
20
3
The input array is
87      100     20      3
The sorted array is
3       20      87      100
```

## PROGRAM-9

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

## ALGORITHM

Algorithm strings

Step 1: Start

Step 2: read string s1 & s2

Step 3: [call function strlength()]

i.e length =strlength

Step 4: Display length 1 and length 2
Step 5: [call function compare string]
if(compare_ string(s1,s2)==0)
print" equal strings"
else
print"unequl strings"
Step 6: call function concatenate
if(concatenate(s1,s2))
Step 7: print"concatenate string"
Step 8: Stop

Algorithm String_length()
Step 1: Start
Step 2: repeate step 2 through while(s1[i]!='\0')
i++
return i
end while
Step 3: Stop

Algorithm String_compare()
Step 1: Start
Step 2: repeatr step 2 trough while(s1[i]==s2[i])
if(s[i]=='\0'|| s2[i]=='\0')
break
i++
end while

Step 3: if(s[i]=='\0' && s2[i]=='\0')
return 0
else
return 1
Step 4: Stop

Algorithm String_concatenate()
Step 1: Start
Step 2: Initilize i=0
Step 3: repeate step 3 through while (s1[i]!='\0')
i++
end while
Step 4: initialize j=0
Step 5: repeate step 5 through while(s2[i]1='\0')
s1[i]=s2[j]
i++ j++
end while
Step 6: s1[i]='\0'
Step 7: Stop

**PROGRAM**

```
#include <stdio.h>
int compare_strings(char [], char []);
void concatenate(char [], char []);
int string_length(char []);

int main()
{
      char s1[100], s2[100];
         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);

}

int string_length(char s1[])
{
          int i = 0;
          while (s1[i] != '\0')
           i++;
```

```
        return i;
}

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

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

**OUTPUT:**

```
Input string1
dsatm
Input string2
dsi
Length of string1:  5
Length of string2:  3
Unequal string
String obtained on concatenation: dstamdsi
```

## PROGRAM-10

**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

Algorithm STUDENT
Step 1: Start
Step 2: create the structure student with fields usn, name and marks and structure variable s[10].
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 & marks for n number of students using structure variable s[i] for i=0 to n-1
Step 6: Display the details
Step 7: repeat for i=0 to n-1
    Compute the sum of students
Step 8: compute the average
Step 9: repeat for i=0 to n-1
    if(s[i].marks >= average)
    printf" total number of students above average"
    else
    printf" total number of students below average"
Step 10: Stop

## PROGRAM

```
#include <stdio.h>
struct student
{
    char usn[50];
    char name[50];
    int marks;
}
int main()
{
    struct student s[10];
    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");
    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");
    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 USN: 11
Enter name: arun
Enter marks: 23
Enter USN: 12
Enter name: ram
Enter marks: 25
Displaying Information:
USN:11
Name: arun
Marks: 23
USN:12
Name: ram
Marks: 25
Average marks: 24
Total No of students above average= 1
Total No of students below average= 1
```

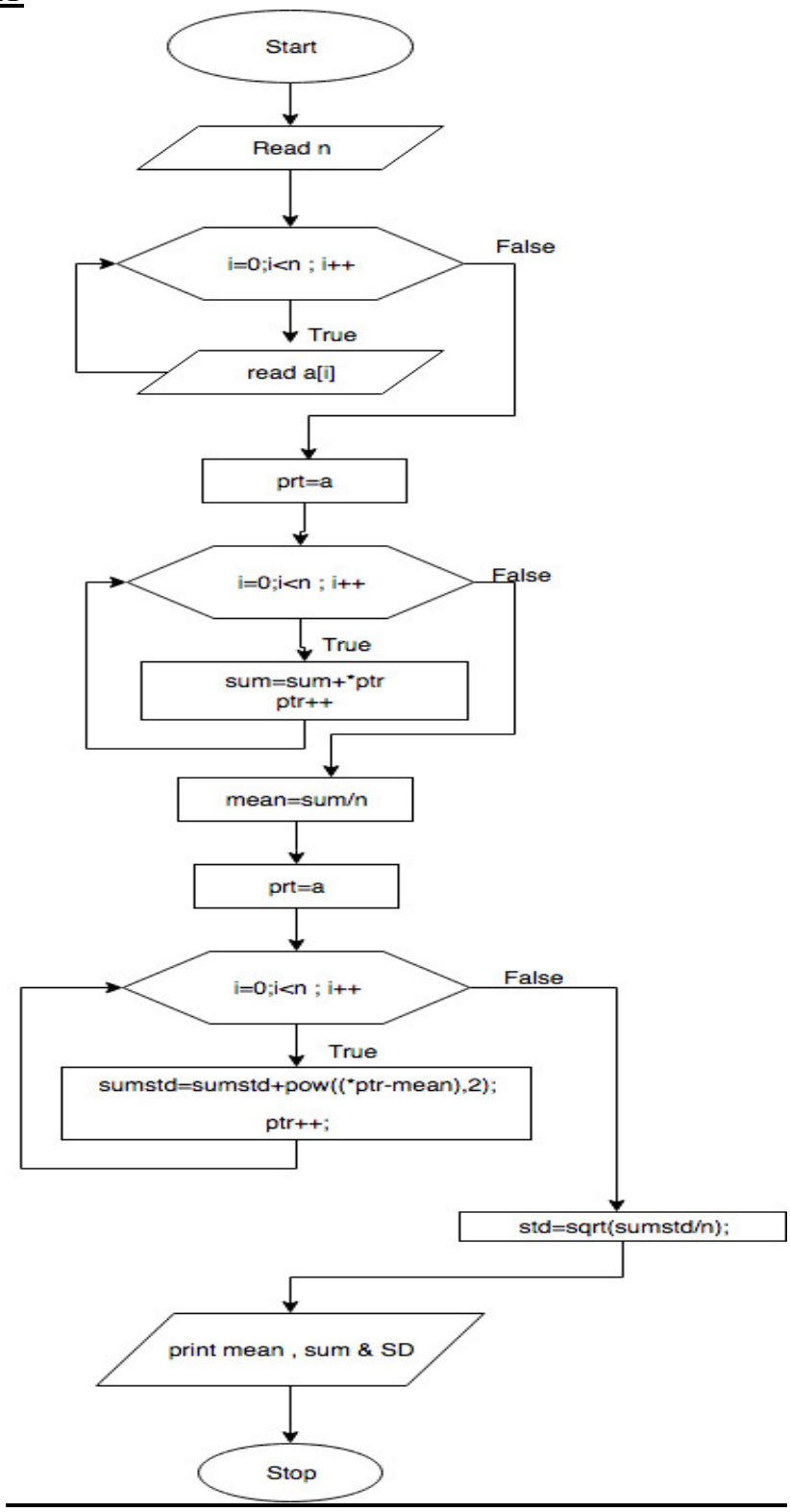
## PROGRAM-11

**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

Algorithm Mean
Step 1: Start
Step 2: read n
Step 3: read for i=0 to n-1
Read a[i]
Step 4: prt=a
Step 5: repeat for i=0 to n-1
sum = sum+*prt
prt++
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(stdsum/n)
Step 10: print mean sun and standard deviation
Step 11: Stop

**FLOWCHART**


Start
Read n
i=0;i<n ; i++
False
True
read a[i]
prt=a
i=0;i<n ; i++
False
True
sum=sum+*ptr
ptr++
mean=sum/n
prt=a
i=0;i<n ; i++
False
True
sumstd=sumstd+pow((*ptr-mean),2);
ptr++;
std=sqrt(sumstd/n);
print mean , sum & SD
Stop

**Program:**

```
#include<stdio.h>
#include<math.h>
int main()
{
	float a[10],*ptr,mean,std,sum=0,sumstd=0;
	int n,i;
	printf("Enter the number of elements\n");
	scanf("%d",&n);
	printf("Enter 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)

}
```

**OUTPUT:**

**.....**

```
Enter the number of elements
5
Enter array elements
1
2
3
4
5
Sum=15.000
Mean=3.000
Standard Deviation =1.414
```

## PROGRAM-12

**Write a C program to copy a text file to another, read both the input file name and target file name.**

### PROGRAM:

```
#include <stdio.h>
#include <stdlib.h> // For exit()

int main()
{
   FILE *fptr1, *fptr2;
   char filename[100], c;

   printf("Enter the filename to open for reading \n");
   scanf("%s", filename);

   // Open one file for reading
   fptr1 = fopen(filename, "r");
   if (fptr1 == NULL)
   {
      printf("Cannot open file %s \n", filename);
      exit(0);
   }

   printf("Enter the filename to open for writing \n");
   scanf("%s", filename);

   // Open another file for writing
   fptr2 = fopen(filename, "w");
   if (fptr2 == NULL)
   {
      printf("Cannot open file %s \n", filename);
      exit(0);
   }

   // Read contents from file
   c = fgetc(fptr1);
   while (c != EOF)
   {
      fputc(c, fptr2);
      c = fgetc(fptr1);
   }

   printf("\nContents copied to %s", filename);

   fclose(fptr1);
   fclose(fptr2);
```

```
    return 0;
}
```

**OUTPUT:**

```
Enter the filename to open for reading
a.txt
Enter the filename to open for writing
b.txt
Contents copied to b.txt
```

## Appendix

## Common 'C' Errors

The most common errors can be broadly classified as follows

1. **Programming errors**
   These errors are generated when typographical errors are made by users.
2. **Compiler errors**
   These errors are detected by the compiler that make the program un-compilable.
3. **Linker error**
   These errors are generated when the executable of the program cannot be generated. This may be due to wrong function prototyping, incorrect header files.
4. **Execution error**
   These errors occur at the time of execution. Looping and arithmetic errors falls under this category.
5. **Logical errors**
   These errors solely depend on the logical thinking of the programmer and are easy to detect if we follow the line of execution and determine why the program takes that path of execution.

### List of some common programming errors

1. Misuse of the Include Guard.
2. Switch statements without break.
3. Wrong usage of postfix and prefix operator (diff between i++ and ++i)
4. Undeclared and Uninitialized Variables and functions.
5. Trying to include "INCORRECT" header function.
6. Returning a value in a void function.
7. Confusing the name of an array with the contents of the first element.
8. 'C' string array Errors - Arr[10] = Arr[0] to Arr[9]. Trying to access Arr[10] element.
9. Using "=" (assignment operator) instead of "= =" (comparison operators) scanf() without '&'
   and wrong format.(IN C)
10. Trying to divide a number by Zero.
11. Poor Loop Exiting comparisons will tend to either loop being not executed at all or goes to
    an infinite loop.
12. Not using string functions and treating the strings are integer. Say trying to compare string by
    (string1= = string2), rather than using strcmp command
13. 'C' String not terminated by '\0'- Null character
14. Mismatched "{" or IF-ELSE statements or for that matter any looping statement.
15. Missing ";" at the end of every statement except control statements and function definition.

### Some Examples of Common Mistakes

**1. Forgetting to put a break in a switch statement**

Remember that C does not break out of a switch statement if a case is encountered. For example:

```
int x = 2;
```

```
switch(x)
{
        case 2:   printf("Two\n");
        case 3:  printf("Three\n");
}
```

prints output:

Two
Three

Put a break to break out of the switch:

```
int x = 2;
switch(x)
{
        case 2:  printf("Two\n");
                 break;
        case 3:  printf("Three\n");
                 break;   /* not necessary, but good if additional cases are added later */
}
```

**2. Using = instead of ==**

C's = operator is used exclusively for assignment and returns the value assigned. The == operator is used exclusively for comparison and returns an integer value (0 for *false*, not 0 for *true*). Because of these return values, the C compiler often does not flag an error when = is used when one really wanted an ==. For example:

```
int x = 5;
if ( x = 6 )
printf("x equals 6\n");
```

This code prints out x equals 6! Why? The assignment inside the if sets x to 6 and returns the value 6 to the if. Since 6 is not 0, this is interpreted as *true*.

One way to have the compiler find this type of error is to put any constants (or any r-value expressions) on the left side. Then if an = is used, it will be an error:

```
if ( 6 = x)
```

**3. scanf() errors**

There are two types of common scanf() errors:

1. Forgetting to put an ampersand (&) on arguments

   scanf() must have the address of the variable to store input into. This means that often the ampersand address operator is required to compute the addresses. Here's an example:

```
scanf("%d", &x);   /* & required to pass address to scanf()    */
scanf("%30s", st); /* NO & here, st itself points to variable! */
```

2. Using the wrong format for operand

   C compilers do *not* check that the correct format is used for arguments of a scanf() call. The most common errors are using the %f format for doubles (which must use the %lf format) and mixing up %c and %s for characters and strings.

**4. Size of arrays**

Arrays in C always start at index 0. This means that an array of 10 integers defined as:

```
int a[10];
```

has valid indices from 0 to 9 *not* 10! It is very common for students go one too far in an array. This can lead to unpredictable behavior of the program.

**5. Integer division**

C uses the / operator for both real and integer division. It is important to understand how C determines which it will do. If both operands are of an integal type, integer division is used, else real division is used. For example:

```
double half = 1/2;
```

This code sets half to 0 not 0.5! Why? Because 1 and 2 are integer constants. To fix this, change at least one of them to a real constant.

```
double half = 1.0/2;
```

If both operands are integer variables and real division is desired, cast one of the variables to double (or float).

```
int x = 5, y = 2;
double d = ((double) x)/y;
```

**6. Loop errors**

In C, a loop repeats the very next statement after the loop statement. The code:

```
int x = 5;
while( x > 0 );
  x--;
```

is an infinite loop. Why? The semicolon after the while defines the statement to repeat as the null statement (which does nothing). Remove the semicolon and the loop works as expected.

Another common loop error is to iterate one too many times or one too few. Check loop conditions carefully!

**7. Not using prototypes**

Prototypes tell the compiler important features of a function: the return type and the parameters of the function. If no prototype is given, the compiler *assumes* that the function returns an int and can take any number of parameters of any type.

One important reason to use prototypes is to let the compiler check for errors in the argument lists of function calls. However, a prototype *must* be used if the function does not return an int. For example, the sqrt() function returns a double, not an int. The following code:

```
double x = sqrt(2);
```

will not work correctly if a prototype:

```
double sqrt(double);
```

does not appear above it. Why? Without a prototype, the C compiler assumes that sqrt() returns an int. Since the returned value is stored in a double variable, the compiler inserts code to convert the value to a double. This conversion is not needed and will result in the wrong value.

The solution to this problem is to include the correct C header file that contains the sqrt() prototype, math.h. For functions you write, you must either place the prototype at the top of the source file or create a header file and include it.

**8. String Errors**

1. Confusing character and string constants

C considers character and string constants as very different things. Character constants are enclosed in *single quotes* and string constants are enclosed in *double quotes*. String constants act as a pointer to the actually string. Consider the following code:

```
char ch = 'A';     /* correct */
char ch = "A";     /* error   */
```

The second line assigns the character variable ch to the address of a string constant. This should generate a compiler error.

2. Comparing strings with ==

Never use the == operator to compare the value of strings! Strings are char arrays. The name of a char array acts like a pointer to the string (just like other types of arrays in C). So what? Consider the following code:

```
char st1[] = "abc";
char st2[] = "abc";
if ( st1 == st2 )
        printf("Yes");
else
        printf("No");
```

This code prints out *No*. Why? Because the == operator is comparing the *pointer values* of st1 and st2, not the data pointed to by them. The correct way to compare string values is to use the strcmp() library function. (Be sure to include string.h) If the if statement above is replaced with the following:

```
if ( strcmp(st1,st2) == 0 )
        printf("Yes");
else
        printf("No");
```

The code will print out *Yes*. For similar reasons, don't use the other relational operators (<,>, *etc*.) with strings either. Use strcmp() here too.

**VIVA QUESTIONS**

1. What is a Computer?
2. What is a compiler?
3. Differentiate object & executable file?
4. What is execution?
5. What is compilation?
6. What are the uses of Internet?
7. Explain different parts of the computer?
8. Define tracing, debugging?
9. What are the differences between RAM and ROM?
10. What are bits? What is a byte?
11. What is booting?
12. What is a source program?
13. Explain the structure of C program with an example.
14. Who is the father of "C" Computer?
15. Steps to execute a program?
16. Which are input & output statements?
17. Explain the different types of IF statements
18. Differentiate between break and continue
19. What are operators? List out different operators?
20. List out fundamental data types in C.
21. What do you mean by type conversions?
22. Why looping is necessary?
23. What is the syntax of a function declaration & function definition?
24. Define recursion & its application?

25. What is an array? How are they declared in 'C'? What are the rules to be followed while using arrays?
26. Explain the single and multi-dimensional arrays?
27. Explain the different categories of functions?
28. What is a string? What is a string manipulation library functions?
29. What are preprocessor directive?
30. List out different header files?
31. What is a parallel program?
32. What is the use of clrscr ()?
33. How to execute a parallel program?
34. What is searching & sorting? Logic of bubble sort, binary search?
35. What is divide & conquer methods?
36. What is memory allocation?
37. What is a pointer?
38. Explain format specifier, escape sequence?
39. Applications of arrays, looping, conditional statements?
40. Can we have main function within a main function?
41. Difference between object oriented & procedure oriented programs?
42. Logic of all programs?
43. Explain flowchart & algorithm?