PROGRAM-1

Aim: *write a java program that simply declares and initializes eight variables, one for each of the eight primitive types and then prints them.*

Description:

The names and values of eight primitive datatypes are:

boolean	: either false or true
char	:16-bit Unicode characters
byte	:8-bit whole numbers : integers ranging from -128 to 127
short	:16-bit whole numbers:integers ranging from -32,768 to 32,767
int	:32-bit whole numbers:integers ranging from -2,147,483,648 to 2,147,483647
long	:64-bit whole numbers:integers ranging from -9223372036854775808 to 9223372036854775807.
float	:32-bit decimal numbers: rationals ranging from $-+1.4*10^{-45}$ to $-+3.4*10^{38}$
double	:64-bit decimal numbers: rationals ranging from $-+4.9 * 10^{-324}$ to $-+1.8* 10^{308}$

The syntax for declaring a variable of any type is

type-name variable_name;

For example, the declaration

int n;

All variables must be declared before they are used. It is usually best to initialize a variable within its declaration. The syntax for that is

***type_name variable_name* = initial_value;**

For example, the declaration

char c= 'R';

Note that literal values of *long* type must end with the letter L and literal values of *float* type end with the letter F.

Also note that values of the type char must be delimited with apostrophes:

char c= 'R';

Program:

```
/* WRITE A JAVA PROGRAM TO DECLARE AND INITIALIZE EIGHT VARIABLES,
ONE FOR EACH OF THE PRIMITIVE TYPES, AND THEN PRINT THEM*/
class PrintTypes
{
        public static void main(String[] args)
        {
                boolean b = false;
                char c = 'R';
                byte j = 127;
                short k = 32767;
                int m = 2147483647;
                long n = 9223372036854775807L;          // 'L' is for "long"
                float x = 3.14159265F;                  // 'F' is for "float"
                double y = 3.1415926535897932 38;
                System.out.println("b="+b);
                System.out.println("c="+c);
                System.out.println("j="+j);
                System.out.println("b="+b);
                System.out.println("k="+k);
                System.out.println("m="+m);
                System.out.println("n="+n);
                System.out.println("x="+x);
                System.out.println("y="+y);
        }
}
```

OUTPUT:

```
student@student-HP-245-G6-Notebook-PC:~/JAVA$ javac PrintTypes.java
student@student-HP-245-G6-Notebook-PC:~/JAVA$ java PrintTypes
b=false
c=R
j=127
b=false
k=32767
m=2147483647
n=9223372036854775807
x=3.1415927
y=3.141592653589793
```

PROGRAM-2

Aim: *write a java program that illustrates how the values of integer variables can be changed with the increment and the decrement operators: ++ and --.and also illustrate the use of the assignment operators += and -=*

Description:

Arithmetic and assignment operators:

An *operator* is a function that has a special symbolic name and is invoked by using that symbol within an expression. Here are some examples of expressions that include java operators:

n = 22	*assignment operator*
n += 22	*assignment operator*
++n	*increment operator*
n / 22	*quotient operator*
n % 22	*remainder operator*

Program:

```
class IncrementDecrement
{
        public static void main(String[] args)
        {
                char c='R';
                byte j=127;
                short k=32767;
                System.out.println("c="+c);
                ++c;
                System.out.println("c="+c);
                ++c;
                System.out.println("c="+c);
                System.out.println("j="+j);
                --j;
                System.out.println("j="+j);
                ++j;
                System.out.println("j="+j);
                ++j;
                System.out.println("j="+j);
                System.out.println("k="+k);
                k-=4;
                System.out.println("k="+k);
                k+=5;
                System.out.println("k="+k);
        }
}
```

OUTPUT:

```
student@student-HP-245-G6-Notebook-PC:~/JAVA$ javac IncrementDecrement.java
student@student-HP-245-G6-Notebook-PC:~/JAVA$ java IncrementDecrement
c=R
c=S
c=T
j=127
j=126
j=127
j=-128
k=32767
k=32763
k=-32768
```

Program-4

Aim: write a java program to find largest among three numbers using ternary operator.

Description:

Ternary operator syntax:

variable = (logicalExpression) ? valueTrueCase : valueFalseCase ;

Java provides a ternary operator **?:** that can be used in place of an if ... else structure in certain cases. Consider an if ... else structure of the following form:

```
if (logicalExpression)
        variable = valueTrueCase ;
else
        variable = valueFalseCase ;
```

Note that in the above statement, if (logicalExpression) evaluates to true, variable = valueTrueCase is executed. Otherwise, variable = valueFalse Case is executed.

In other words, the purpose of the above if structure is to assign one value or another to the same variable based on the truth or falsehood of a logical expression.

In such cases, you can replace the above if ... else structure by the following semantically equivalent Java statement:

variable = (logicalExpression) ? valueTrueCase : valueFalseCase ;

Program:

```
import java.util.Scanner;
class LargestTernary
{
        public static void main(String[] args)
        {
                int d;
                Scanner s=new Scanner(System.in);
                System.out.println("Enter all three numbers");
                int a=s.nextInt();
                int b=s.nextInt();
                int c=s.nextInt();
                d=c>(a>b?a:b)?c:(a>b?a:b);
                System.out.println("Largest Number:"+d);
        }
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ javac LargestTernary.java
student@student-HP-245-G6-Notebook-PC:~$ java LargestTernary
Enter all three numbers
56
33
88
Largest Number:88
```

PROGRAM-5

Aim: *Write a java program to find the roots of Quadratic Equation.*

Description:

- The Math class in ***java.lang*** package contains class methods for commonly used mathematical functions.
- Most methods of the Math class are static and therefore there is no need to instantiate any new object of the Math class before using it.
- Most Math class methods generate a **double** type.
- Math class methods include

  abs(a): Returns the absolute value of a.
  ceil(a): Returns the smallest whole number greater than a.
  floor(a): Returns the largest whole number less than a.
  max(a,b): Returns the larger of a and b.
  min(a,b): Returns the smaller of a and b.
  pow(a,b): Returns the number a raised to power b.
  random(): Generates a random number less than or equal to 0.0 and less than 1.0.
  sqrt(a): returns the square root of a.

Program:

```
//to find root of quadratic equation
import java.util.Scanner;
class Quad
{
        public static void main(String[] args)
        {
                double root1,root2,d;
                Scanner s= new Scanner(System.in);
                System.out.println("Enter a,b,c values");
                int a= s.nextInt();
                int b= s.nextInt();
                int c= s.nextInt();
                d=b*b-4*a*c;
                if(d>0)
                {
                        System.out.println("ROOTS ARE REAL AND UNEQUAL");
                        root1=(-b+Math.sqrt(d))/(2*a);
                        root2=(-b-Math.sqrt(d))/(2*a);
                        System.out.println("First root is:"+root1);
                        System.out.println("Second root is:"+root2);
                }
                else if(d==0)
                {
                        System.out.println("ROOTS ARE REAL AND EQUAL");
                        root1=(-b+Math.sqrt(d))/(2*a);
                        root2=(-b-Math.sqrt(d))/(2*a);
                        System.out.println("First root is:"+root1);
                        System.out.println("Second root is:"+root2);
                }
                else
                {
                        System.out.println("Roots are imaginary");
                }
        }
}
```

Output:

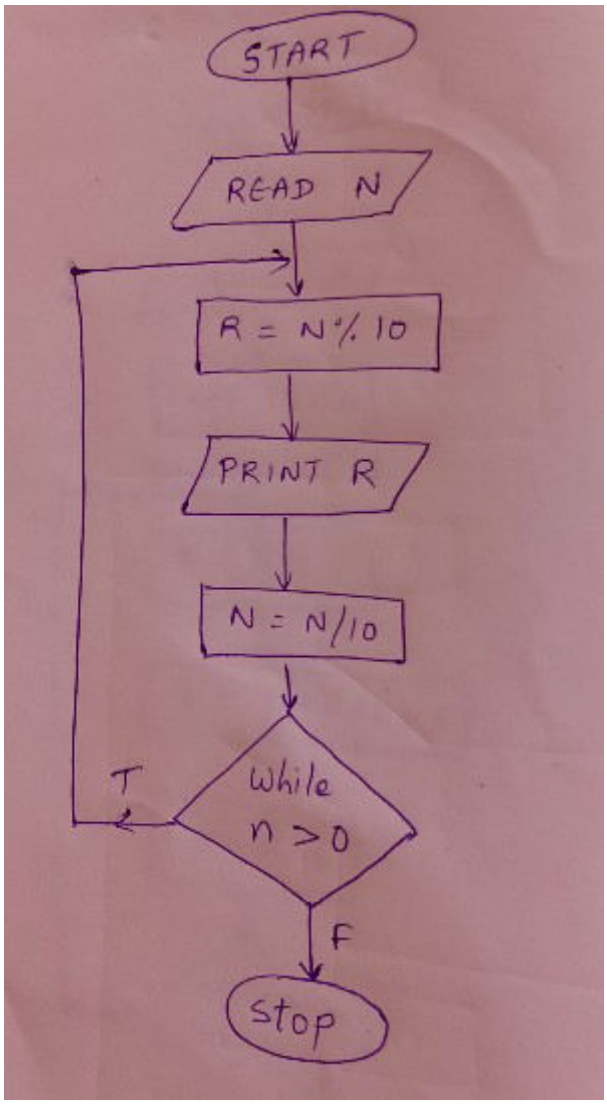
```
student@student-HP-245-G6-Notebook-PC:~$ javac Quad.java
student@student-HP-245-G6-Notebook-PC:~$ java Quad
```

```
Enter a,b,c values
2
3
1
ROOTS ARE REAL AND UNEQUAL
First root is:-0.5
Second root is:-1.0
```

PROGRAM-6

AIM: Write a java program to reverse a number.

FLOWCHART:



PROGRAM:

```
//to reverse a number
import java.util.Scanner;
class Reverse
{
        public static void main(String[] args)
        {
                int r;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter a number");
                int n=sc.nextInt();
                while(n>0)
                {
                        r=n%10;
                        System.out.print(r);
                        n=n/10;
                }

        }
}
```

OUTPUT:

```
student@student-HP-245-G6-Notebook-PC:~$ javac Reverse.java
student@student-HP-245-G6-Notebook-PC:~$ java Reverse
Enter a number
3456
6543
```

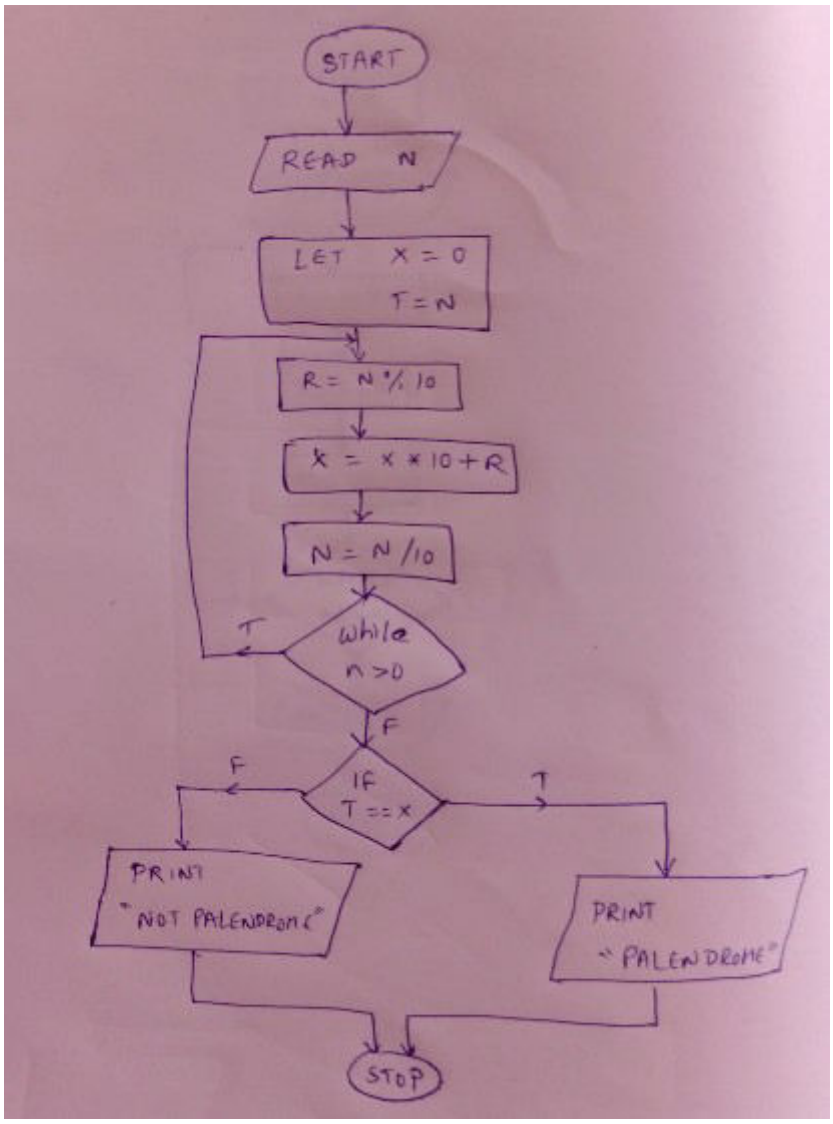
PROGRAM-7

AIM: *Write a java program to check the given number is Palindrome or not.*

DESCRIPTION:

The digits of a palindromic number are the same read backwards as forwards, for example, 91019. For example, 191 and 313 are palindromes

FLOWCHART:



Program:

```
//to check whether a number is palendrome or not
import java.util.Scanner;
class Palendrome
{
        public static void main(String[] args)
        {
                int rev=0,r,t;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter a number");
                int n=sc.nextInt();
                t=n;
                while(n>0)
                {
                        r=n%10;
                        rev=rev*10+r;
                        n=n/10;
                }
                if(t==rev)
                {
                        System.out.println("it is a palendrome");
                }
                else
                {
                        System.out.println("it is not a palendrome");
                }

        }
}
```

OUTPUT:

student@student-HP-245-G6-Notebook-PC:~$ **javac Palendrome.java**
student@student-HP-245-G6-Notebook-PC:~$ **java Palendrome**
Enter a number
131
it is a palendrome

PROGRAM-8

AIM: *Write a java program to check whether a number is armstrong or not.*

DESCRIPTION:

Armstrong number is a number that is equal to the sum of cubes of its digits. For example 0, 1, 153, 370, 371 and 407 are the Armstrong numbers.

Let's try to understand why **153** is an Armstrong number.

153 = (1*1*1)+(5*5*5)+(3*3*3)

= 1+125+27

= 153

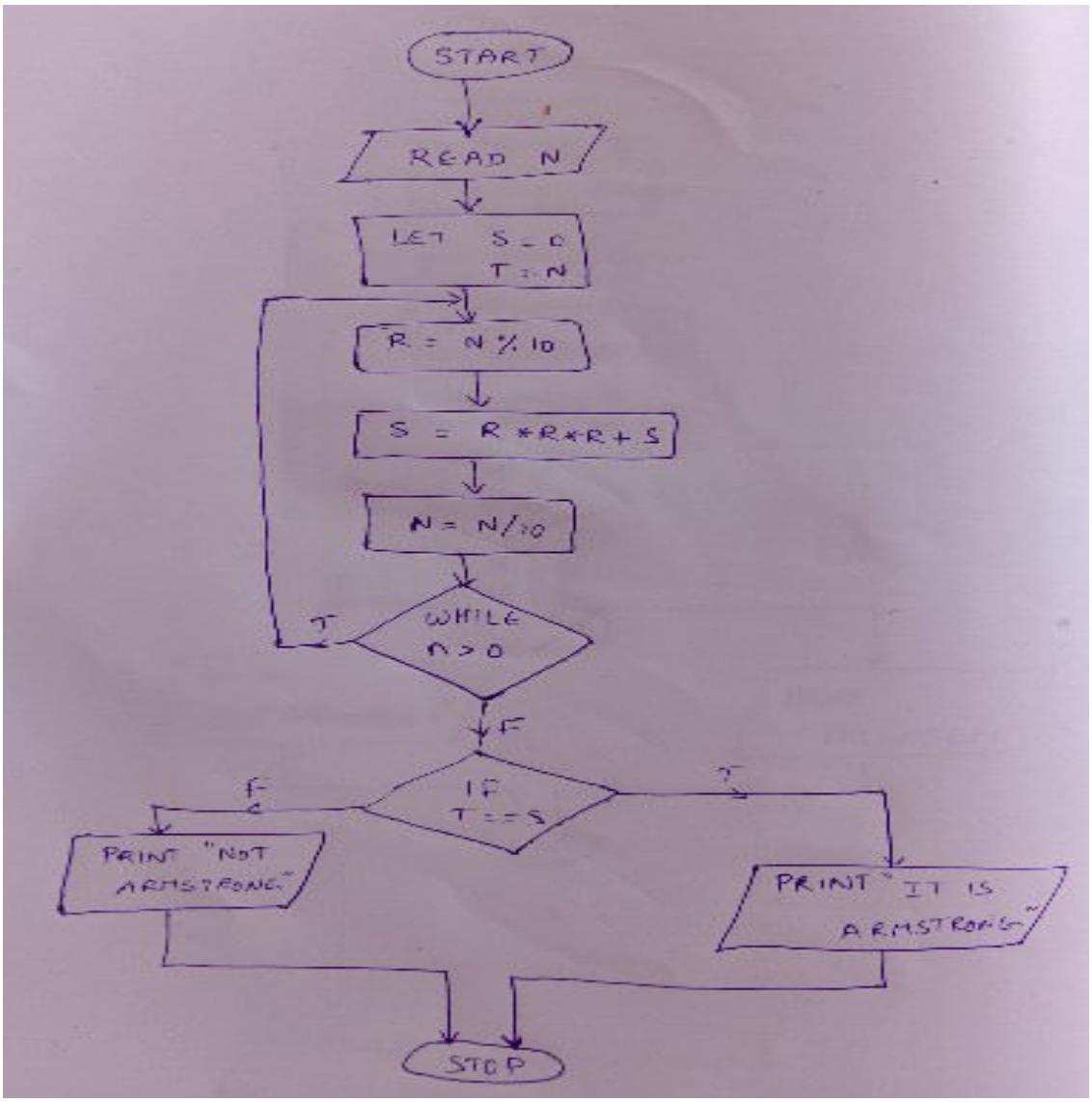
Let's try to understand why **371** is an Armstrong number.

371 = (3*3*3)+(7*7*7)+(1*1*1)

= 27+343+1

= 371

FLOWCHART:



PROGRAM:

```
//to check whether a number is armstrong or not
import java.util.Scanner;
class Armstrong
{
        public static void main(String[] args)
        {
                int s=0,r,t;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter a number");
                int n=sc.nextInt();
                t=n;
                while(n>0)
                {
                        r=n%10;
                        s=r*r*r+s;
                        n=n/10;
                }
                if(t==s)
                {
                        System.out.println("it is an armstrong number");
                }
                else
                {
                        System.out.println("it is not an armstrong number");
                }

        }
}
```

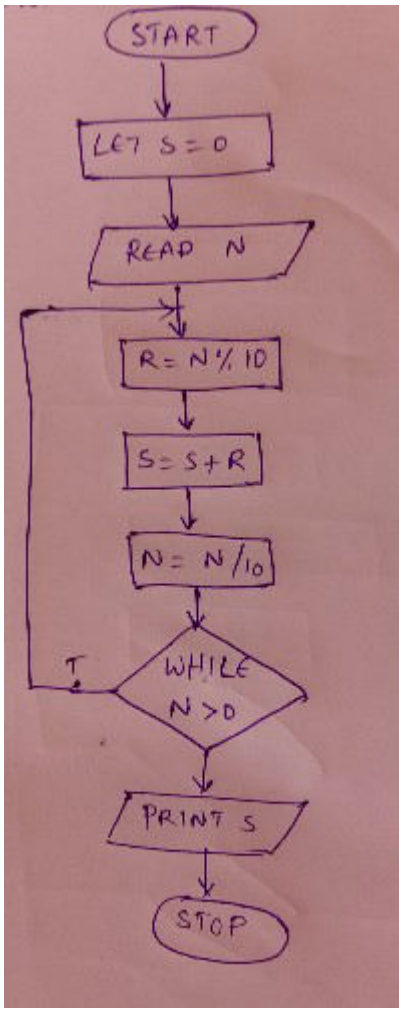
OUTPUT:

student@student-HP-245-G6-Notebook-PC:~$ **javac Armstrong.java**
student@student-HP-245-G6-Notebook-PC:~$ **java Armstrong**
Enter a number
153
it is an armstrong number

PROGRAM-9

AIM: *Write a java program to print sum of digits of a number.*

FLOWCHART:



PROGRAM:

```
// JAVA PROGRAM TO PRINT SUM OF DIGITS OF THE NUMBER
import java.util.Scanner;
class Sum
{
        public static void main(String[] args)
        {
                int s=0,r;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter a number");
                int n=sc.nextInt();
                while(n>0)
                {
                        r=n%10;
                        s=s+r;
                        n=n/10;
                }
                System.out.println("sum of the digits of the number:"+s);

        }
}
```

OUTPUT:

student@student-HP-245-G6-Notebook-PC:~$ **javac Sum.java**
student@student-HP-245-G6-Notebook-PC:~$ **java Sum**
Enter a number
345
sum of the digits of the number:12

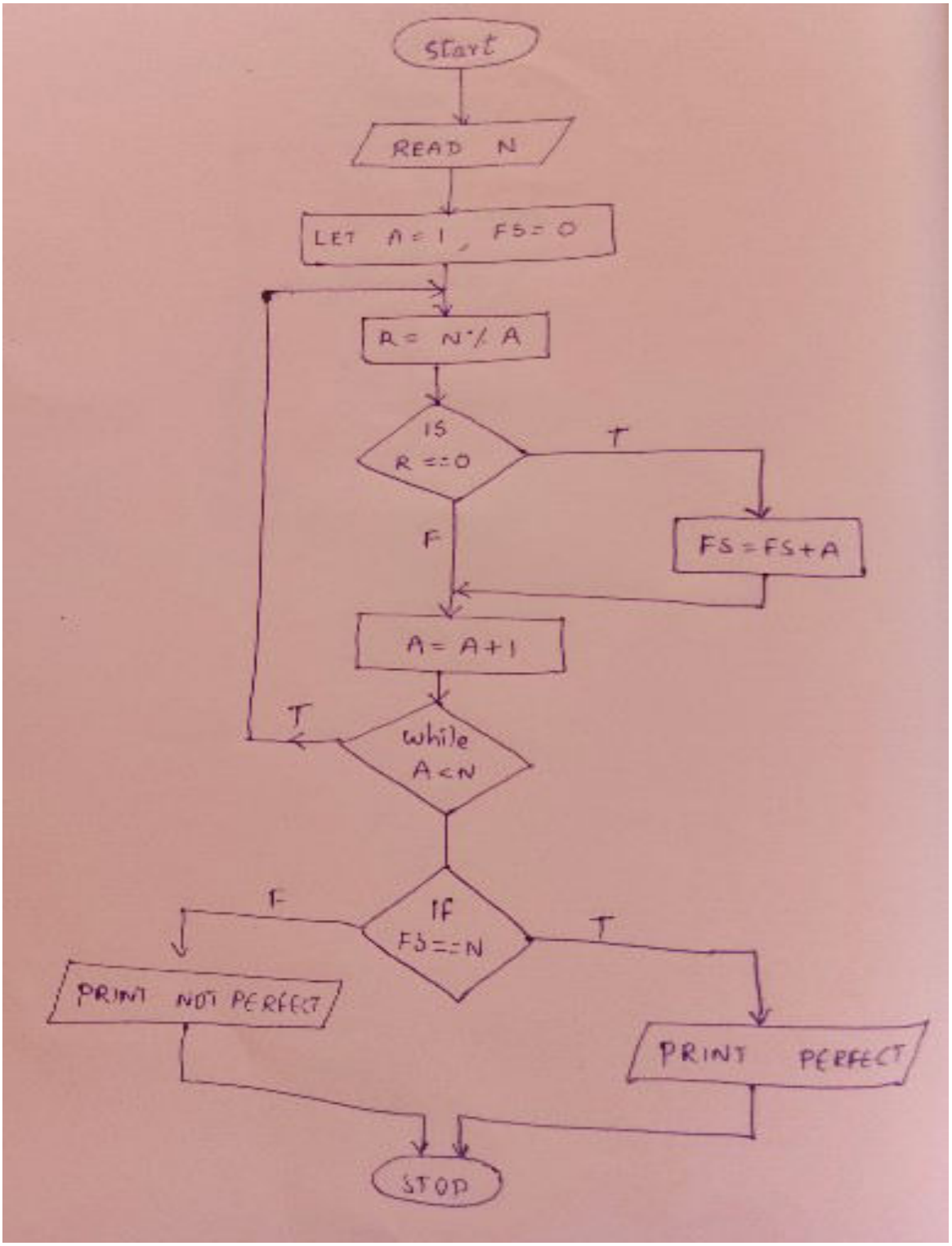
Program-10

Aim: *Write a java program to check whether a number is perfect or not.*

Description:

In number theory, a **perfect number** is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. For example, 6 has divisors 1, 2 and 3 (excluding itself), and 1 + 2 + 3 = 6, so 6 is a perfect number.

Flowchart:



Program:

```
//to check whether a number is perfect or not
import java.util.Scanner;
class Perfect
{
        public static void main(String[] args)
        {
                int fs=0,a=1,r;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter a number");
                int n=sc.nextInt();
                while(a<n)
                {
                        r=n%a;
                        if(r==0)
                        {
                                fs=fs+a;
                        }
                        a=a+1;
                }

                if(fs==n)
                {
                        System.out.println("it is perfect number");
                }
                else
                {
                        System.out.println("it is not perfect number");
                }

        }
}
```

Output:

student@student-HP-245-G6-Notebook-PC:~$ **javac Perfect.java**
student@student-HP-245-G6-Notebook-PC:~$ **java Perfect**
Enter a number
28
it is perfect number

PROGRAM-11

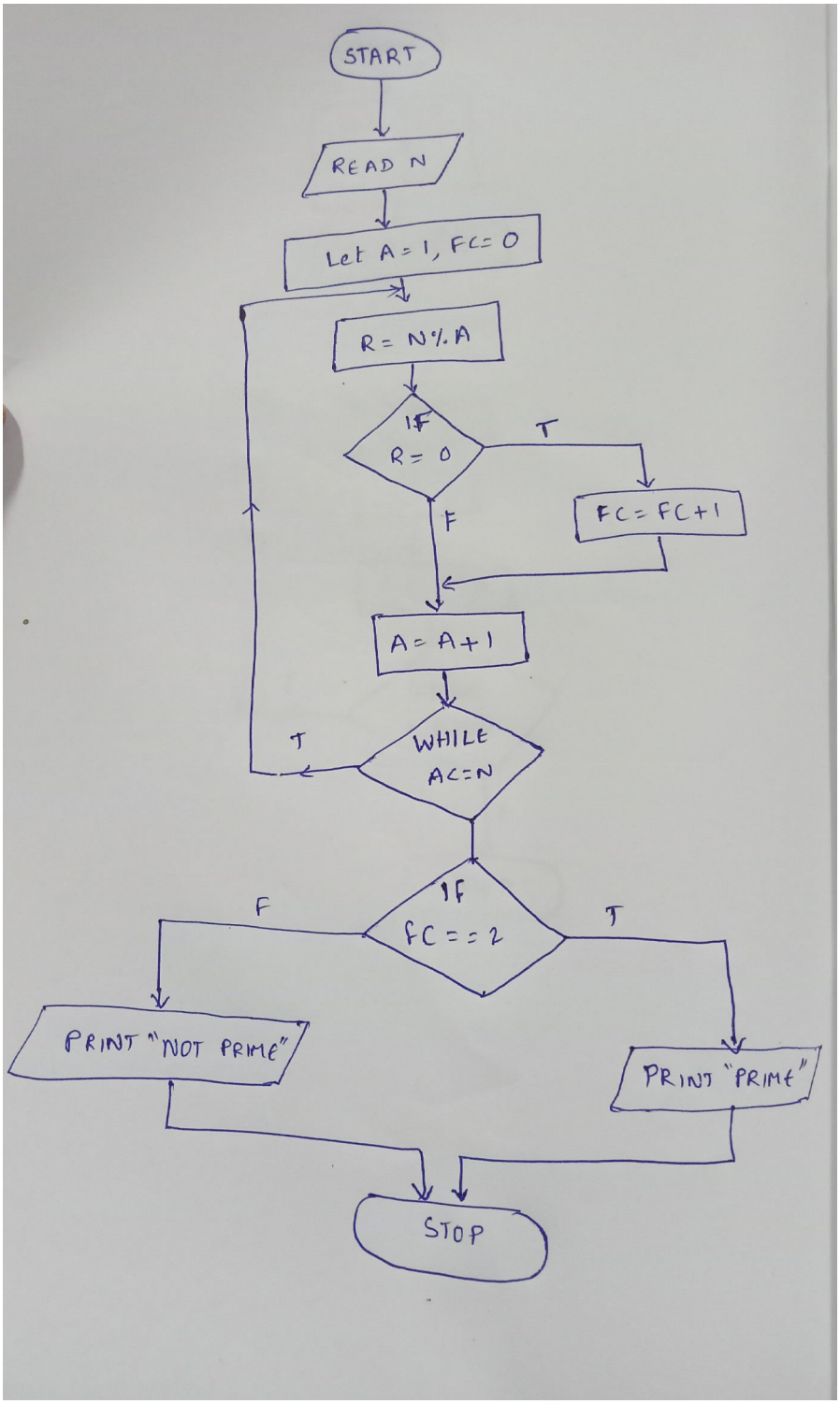
AIM: *Write a java program to check the given number is Prime or not.*

DESCRIPTION:

A prime number is a number which is divisible by only two numbers: 1 and itself. So, if any number is divisible by any other number, it is not a prime number.

Here are the first few **prime numbers**: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

FLOWCHART:



PROGRAM:

```
//Write a java program to check the given number is prime or not
import java.util.Scanner;
class Primenum
{
        public static void main(String[] args)
        {
                int n,i,fcount=0;
                System.out.println("Enter the number to evaluate for prime number.");
                Scanner sc=new Scanner(System.in);
                n=sc.nextInt();
                for(i=1;i<=n;i++)
                {
                        if(n%i==0)
                        {
                                fcount++;
                        }
                }
                if(fcount==2)
                        System.out.println("it is a prime number");
                else
                        System.out.println("it is not a prime number");
        }
}
```

OUTPUT:

student@student-HP-245-G6-Notebook-PC:~$ **javac Primenum.java**
student@student-HP-245-G6-Notebook-PC:~$ **java Primenum**
Enter the number to evaluate for prime number.
29
it is a prime number

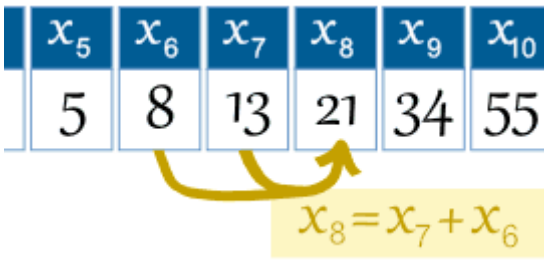
PROGRAM-12

AIM: *Write a java program to generate Fibonacci series of the given number.*

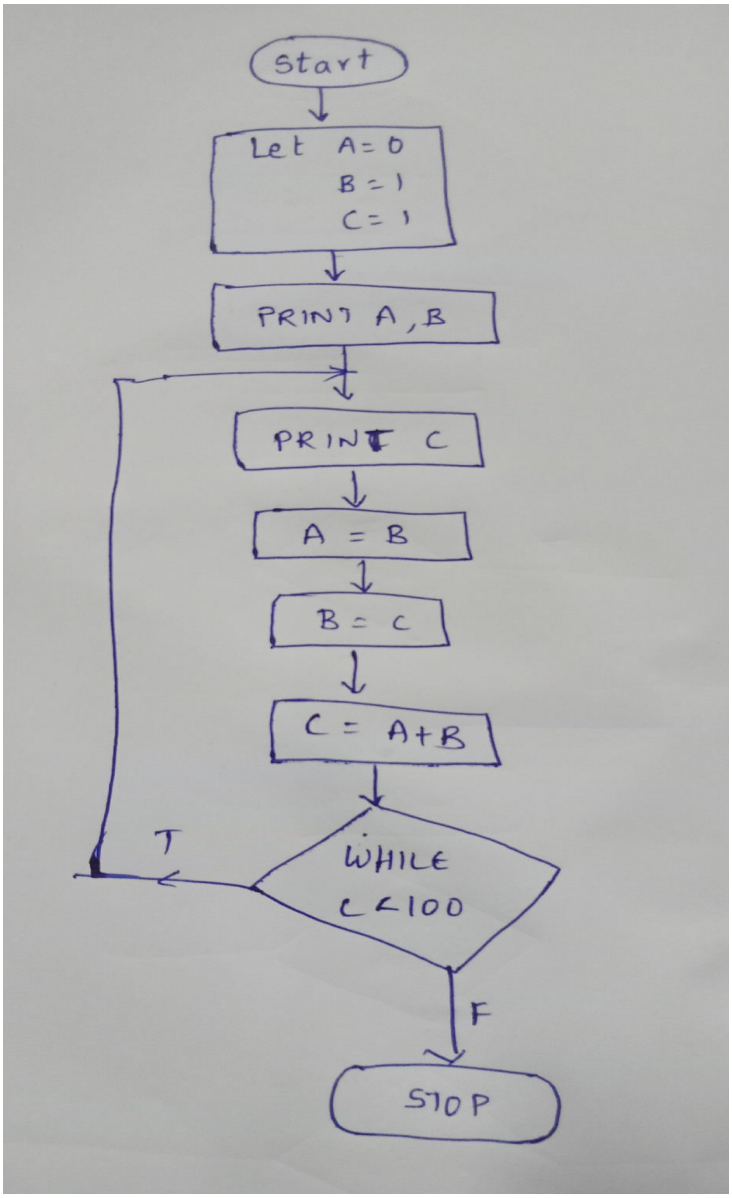
Description:

The Fibonacci sequence is one of the most famous formulas in mathematics.

Each number in the sequence is the sum of the two numbers that precede it. So, the sequence goes: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, and so on. The mathematical equation describing it is $X_{n+2}= X_{n+1} + X_n$



FLOWCHART:



PROGRAM:

```
//Java program to generate Fibonacci series of the given number
import java.util.Scanner;
class Fib
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                int i,a=0,b=1,c=0,n;
                System.out.print("Enter value of n:");
                n=sc.nextInt();
                System.out.println(a);
                System.out.println(""+b);
                i=2;
                while(i<=n)
                {
                        c=a+b;
                        System.out.println(""+c);
                        a=b;
                        b=c;
                        i++;
                }
        }
}
```

OUTPUT:

student@student-HP-245-G6-Notebook-PC:~$ **javac Fib.java**
student@student-HP-245-G6-Notebook-PC:~$ **java Fib**
Enter value of n:10
0
1
1
2
3
5
8
13
21
34
55

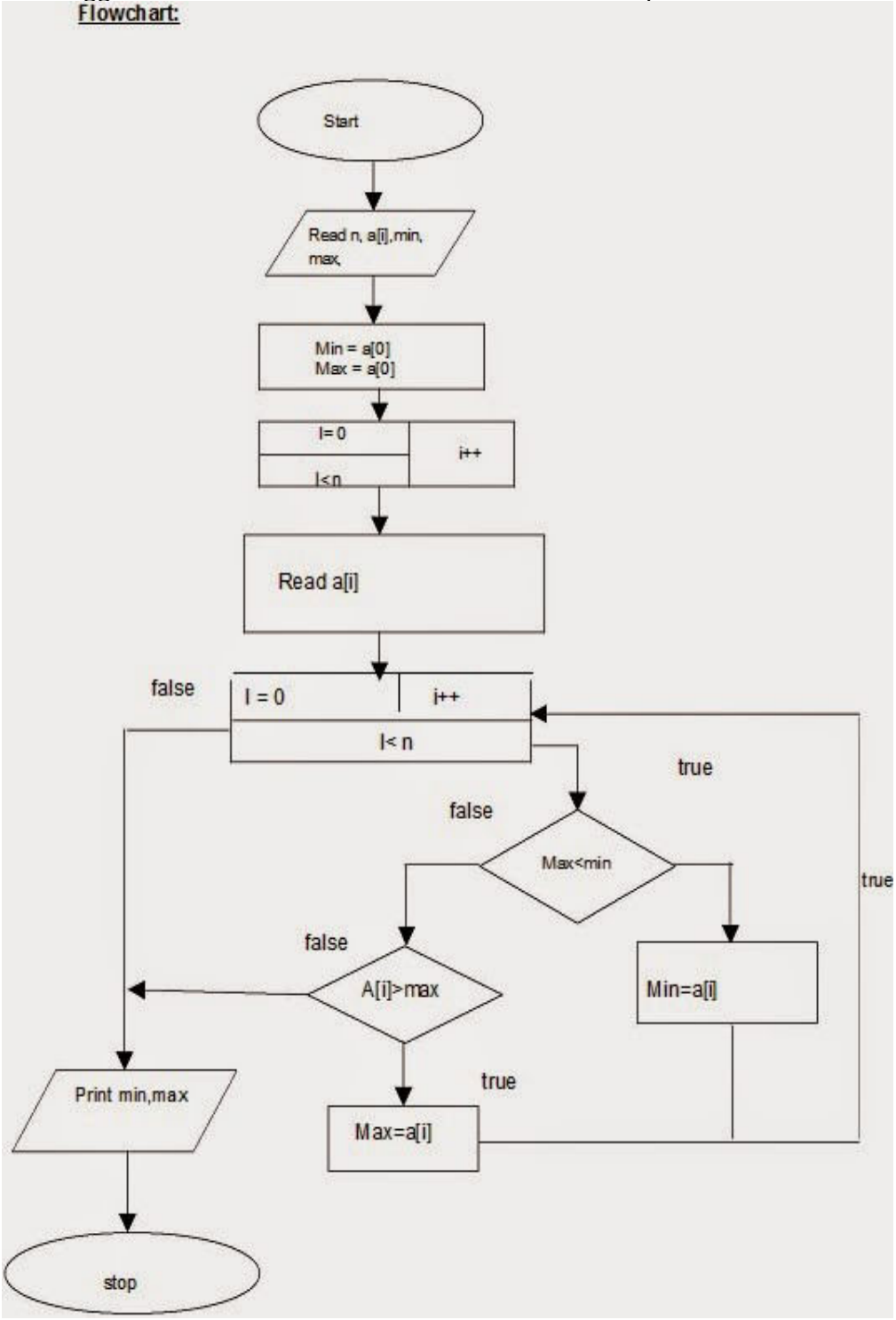
PROGRAM 2.1

AIM: *Write a java program to find Largest and smallest number in an array*

DESCRIPTION:

This program Finds the Largest Number and the Smallest Number present in the Array Variable. It uses for loop to iterate the array upto the end of its index value and if conditions evaluates the biggest and smallest numbers and that store in the specified variable.

Flowchart:



PROGRAM:

```
//program to find largest and smallest number in an array
class Test21
{
        public static void main(String[] args)
        {
                int numbers[]=new int[]{32,43,53,54,32,65,63,98,43,23};
                //assign first element of an array to largest and smallest
                int smallest=numbers[0];
                int largest=numbers[0];
                for(int i=1;i<numbers.length;i++)
                {
                        if(numbers[i]>largest)
                                largest=numbers[i];
                        else if(numbers[i]<smallest)
                                smallest=numbers[i];
                }
                System.out.println("Largest number is"+largest);
                System.out.println("Smallest number is"+smallest);
        }
}
```

OUTPUT:

student@student-HP-245-G6-Notebook-PC:~$ **javac Test21.java**
student@student-HP-245-G6-Notebook-PC:~$ **java Test21**
Largest number is98
Smallest number is23

PROGRAM 2.2

AIM: *Write a java program for linear search using arrays.*

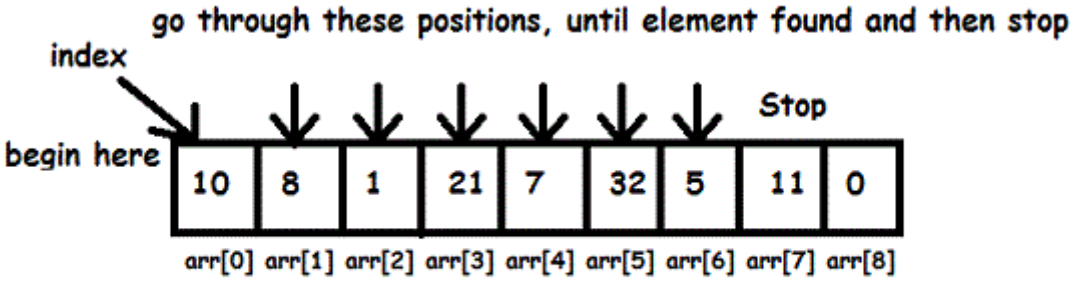
DESCRIPTION:

A **linear search** is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.This method is referred as serial or sequential search.

Best case: If the first element itself is the required element, then we would have spent one comparison. Thus, the best case time complexity can be said as **1.**

Worst case: If none of the element is same as the search element, we would have spent **n** comparisions .Thus,the worst case time complexity can be said as **n.**

Example 1:



Element to search : 5

Example 2:

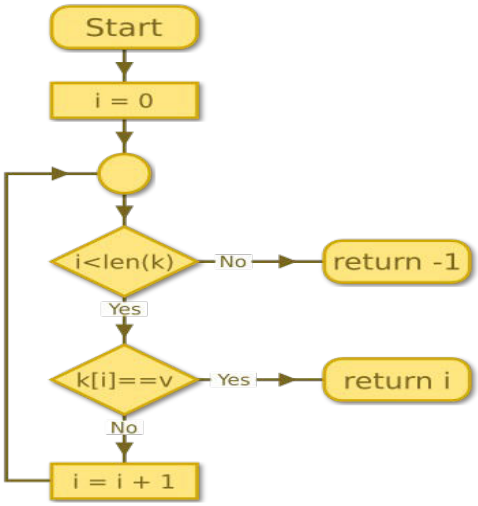
Array **numlist** contains

17	23	5	11	2	29	3

Searching for the value 11, Linear search examines 17,23,5 and 11

Searching for the value 7, Linear search examines 17,23,5,11,2,29 and 3

FLOWCHART:



PROGRAM:

```
import java.util.Scanner;
class LinearSearch
{
        public static void main(String[] args)
        {
                int search,i;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter number of elements");
                int n=sc.nextInt();
                int array[]=new int[n];
                System.out.println("Enter" + n+"integers");
                for(i=0;i<n;i++)
                {
                        array[i]=sc.nextInt();
                }
                System.out.println("Enter value to find");
                search=sc.nextInt();
                for(i=0;i<n;i++)
                {
                        if(array[i]==search)
                        {
                                System.out.println(search+"is present at location"+(i+1)+".");
                                break;
                        }
                }
                if(i==n)
                        System.out.println(search+"is  not present in array");
        }
}
```

student@student-HP-245-G6-Notebook-PC:~$ **javac LinearSearch.java**
student@student-HP-245-G6-Notebook-PC:~$ **java LinearSearch**
Enter number of elements
4
Enter4integers
10
20
30
40
Enter value to find
20
20is present at location2.

PROGRAM 2.3

AIM: Write a java program to add two matrices

DESCRIPTION: The program takes the two matrixes of same size and performs the addition

Algorithm:

Step 1: start

Step 2: read the size of matrices A,B – m,n

Step 3: read the elements of matrix A

Step 4: read the elements of matrix B

Step 5: perform the addition operation

Step 6: print sum of matrices A and B

Step 7: Stop

PROGRAM:

```
//Java program to add two matrices
import java.util.Scanner;
class AddTwoMatrix
{
        public static void main(String[] args)
        {
                int c,d;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the number of rows and columns of matrix");
                int m=sc.nextInt();
                int n=sc.nextInt();
                int first[][]=new int[m][n];
                int second[][]=new int[m][n];
                int sum[][]=new int[m][n];
                System.out.println("Enter the elements of first matrix");
                for(c=0;c<m;c++)
                        for(d=0;d<n;d++)
                                first[c][d]=sc.nextInt();
                System.out.println("Enter the elements of second matrix");
                for(c=0;c<m;c++)
                        for(d=0;d<n;d++)
                                second[c][d]=sc.nextInt();
                for(c=0;c<m;c++)
                        for(d=0;d<n;d++)
                                sum[c][d]=first[c][d]+second[c][d];
                System.out.println("Sum of entered matrices:");
                for(c=0;c<m;c++)
                {
                        for(d=0;d<n;d++)
                                System.out.print(sum[c][d]+"\t");
                        System.out.println();
                }
        }
}
```

OUTPUT

student@student-HP-245-G6-Notebook-PC:~$ **javac AddTwoMatrix.java**
student@student-HP-245-G6-Notebook-PC:~$ **java AddTwoMatrix**

```
Enter the number of rows and columns of matrix
2 2
Enter the elements of first matrix
1 2
3 4
Enter the elements of second matrix
4 3
2 1
Sum of entered matrices:
5       5
5       5
```

PROGRAM 2.4

AIM: *Write a java program to multiply two matrices. Before multiplication matrices are checked whether they can be multiplied or not*

DESCRIPTION:

For multiplication to be defined, the "inner" numbers must match. The result will be determined by "outer" numbers.

$$\begin{bmatrix} 2 & 1 & 4 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 6 & 3 & -1 & 0 \\ 1 & 1 & 0 & 4 \\ -2 & 5 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 27 & -2 & 12 \\ -1 & 6 & 0 & 6 \end{bmatrix}$$

2 x 3 3 x 4 2 x 4

match

result

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

2 x 4 4 x 3 2 x 3

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

Algorithm

- **STEP 1:** START
- **STEP 2:** DEFINE row1, col1, row2, col2
- **STEP 3:** INITIALIZE matrix a[][] ={{1,3,2},{3,1,1}, {1,2,2}}
- **STEP 4:** INITIALIZE matrix b[][] ={{2,1,1},{1,0,1}, {1,3,1}}
- **STEP 5:** row1 = a.length
- **STEP 6:** col1 = a[0].length
- **STEP 7:** row2 =b.length
- **STEP 8:** row2 = b[0].length
- **STEP 9:** if(col1!=row2)
  PRINT "Matrices cannot be multiplied"
  else
  Go to step 10;
- **STEP 10:** prod[][] = [row1][col2]
- **STEP 11:** REPEAT STEP 12 to STEP 14 UNTIL i<row1
  //for(i=0; i<row1; i++)

- **STEP 12:** REPEAT STEP 13 to STEP 14 UNTIL j<col2 // for(j=0; j<col2; j++)
  If(j>i) then PRINT 0 else PRINT a[i][j]
- **STEP 13:** REPEAT STEP 14 UNTIL k<row2 // for(k=0; k<row2; k++)
- **STEP 14:** prod[i][j] = prod[i][j] + a[i][k]*b[k][j]
- **STEP 15:** REPEAT STEP 16 to STEP 18 UNTIL i<row1
- **STEP 16:** REPEAT STEP 17 UNTIL j<col2
- **STEP 17:** PRINT prod[i][j]
- **STEP 18:** PRINT new line
- **STEP 19:** END

PROGRAM:

```
//Java program to multiply two matrices
import java.util.Scanner;
class MatrixMultiplication
{       public static void main(String[] args)
        {       int sum=0,m,n,p,q,c,d,k;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the number of rows and columns of first matrix");
                m=sc.nextInt();
                n=sc.nextInt();
                int first[][]=new int[m][n];
                System.out.println("Enter the elements of first matrix");
                for(c=0;c<m;c++)
                        for(d=0;d<n;d++)
                                first[c][d]=sc.nextInt();
                System.out.println("Enter the elements of rows and columns of second matrix");
                p=sc.nextInt();
                q=sc.nextInt();
                 if(n!=p)
                        System.out.println("Matrices with entered orders cant be multiplied with each other");
                 else
                 {       int second[][]=new int[p][q];
                         int multiply[][]=new int[m][q];
                         System.out.println("Enter the elements of second matrix");
                         for(c=0;c<p;c++)
                                 for(d=0;d<q;d++)
                                         second[c][d]=sc.nextInt();
                         for(c=0;c<m;c++)
                         {       for(d=0;d<q;d++)
                                 {       for(k=0;k<p;k++)
                                         {
                                                 sum=sum+first[c][k]*second[k][d];
                                         }
                                 multiply[c][d]=sum;
                                 sum=0;
                                 }
                         }
                         System.out.println("Product of entered matrices:");
                         for(c=0;c<m;c++)
                         {
                                 for(d=0;d<q;d++)
                                         System.out.print(multiply[c][d]+"\t");
                                 System.out.println("\n");
                         }

                 }
        }
}
```

OUTPUT

student@student-HP-245-G6-Notebook-PC:~$ **javac MatrixMultiplication.java**
student@student-HP-245-G6-Notebook-PC:~$ **java MatrixMultiplication**
Enter the number of rows and columns of first matrix
3 3
Enter the elements of first matrix
1 2 3
4 5 6

7 8 9
Enter the elements of rows and columns of second matrix
3 3
Enter the elements of second matrix
9 8 7
6 5 4
3 2 1
Product of entered matrices:
30 24 18

84 69 54

138 114 90