Welcome to the World Of ‘C’

- ❑ **TEXT BOOK:**

- Programming in ANCI C- Third edition
  Ashok M Kamthane, Amit Ashok Kamthane
- Computer Fundamentals- Fourth Edition
  P K Sinha , Preethi Sinha

- ❑ **Reference Books:**

- Programming in ANSI C-Seventh Edition
  E. Balaguruswamy

Importance of Subject

- C is the base language of any other programming language.
- To be a Good Programmer one must know fundamentals of C programming language.

History of ‘C’

- Root of the modern C language is **ALGOL** 1960.
  - It is first computer language to use a block structure.
  - It gave the concept of structured programming.
- In 1967, Martin Richards developed a language, **BCPL** (Basic Combined Programming Language)

- ❑ In 1970, Ken Thompson created a language called as ‘B’.
  - It was used to create the early version of **Unix.**

- ❑ In 1972,by Dennis Ritchie introduced new language called as ‘C’ .
  - Developed in Bell laboratories for Army needs

1972	Traditional C	Dennis Ritchie
1978	K&R C	Kernighan &Ritchie
1989	ANSI C	ANSI Committee
1990	ANSI/ISO C	ISO Committee

Features Of ‘C’

- It is a robust language- Complex programs, powerful language
- Programs written in ‘C’ are efficient and fast- Because of variety of data types and powerful operators
- Highly Portable. (related to OS)
- Well suited for structured programming.
- Ability to extend itself.

Program & Programming Language

- Program:-

  A Set of instructions to solve a problem

- Programming language

  A specific manner of writing a program with some predefined rules, symbols & their use as a part of language.

  i.e. Pascal, C, C++, VC++, JAVA, VB.

Syntax and Semantics

- Syntax: Grammar
- Semantics: Meaning

A program should have correct syntax and semantics.

Example:

A boy play with the ball. (syntax wrong)
A boy plays with the ball. (syntax correct, semantics correct)

A ball plays with the boy. (syntax correct, semantics incorrect)

A small C Program

```
# include<stdio.h>
# include<conio.h>
void main()
• {
/* print a sentence*/
printf("welcome to BPC College");
getch();
}
```

Header files

Main function

Comment

Function Body

How to run a program?

- Program is written in source code(human readable form)
- Compilation- A compiler translates the source code to object code(machine code)
- Execution
- Output

C Compiler

- checks for syntax errors if any
- on success <u>coverts</u> ‘C source code into object code form – which is nearer to machine code…

Characteristics of a good program

Some important characteristic of well written computer programs are:

- **Integrity:** This refers to the accuracy of calculation. It is clear that all other program enhancement will be meaning less, if the calculations are not carried out correctly.
- **Clarity:** This refer to the overall readability of this program . If a program is clearly written, it should be possible for another programmer to follow the program logic without any effort.

Characteristics of a program

- **Simplicity**: The clarity and accuracy of a program are usually enhanced by keeping things as simple as possible.
- **Efficiency**: This is concerned with speed and efficient memory utilization .They are at the expense of clarity, simplicity and accuracy .
- **Generality**: The program should be as general as possible.

Characteristics of a good program

- Modularity: Many programs can be broken down into a series of identifiable sub tasks. It is a good, programming practice to implement each of these sub tasks as a separate program module. In C, such modules are written as functions.

C Program- To Add Two Numbers

```
# include<stdio.h>
#include<conio.h>
void main()
{/* to add two nos.*/  //comment
int a, b, sum;             //declaration
printf(“enter two nos.\n”);   //output statement
scanf(“%d%d”, &a,&b);      //input statement
sum=a+b;
printf(“the sum is %d”, sum);
getch();
}
```

Basic structure of 'C'

1) <u>Documentation Section</u> :-

   It has set of comment lines(name of program, author details).

   What is Comment line??

- To guide a programmer. To write a note for function,operation,logic in between a program.
- Non-executable statement.
- Can't be nested.

  e.g:- /* Hello /* abc */ Hi */

  ERROR.

2) Link Section :-

It provides instructions to the compiler to link function from the system library.

# include Directive:-

- To access the functions which are stored in the library, it is necessary to tell the compiler , about the file to be accessed.

  Syntax:-

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

- stdio.h is header file.

3) Definition Section

- It defines all symbolic constants.
- #define instuction defines value to a symbolic constant.
- #define:-

▪ It is a preprocessor compiler directive, not a statement.

▪ Therefore it should not end with a semicolon.

▪ Generally written in uppercase.

4) Global Declaration Section

- Some variables that are used in more than one function, such variables (global variables) are declared in the global declaration section.

- It also declares all the user-defined functions.

5) <u>Main() function section:</u>

- Every ‘C’ program must have one main() function section.
- It contains two parts

1) <u>Declaration part</u>:

- It declares all variables used in the executable part.

2) <u>Executable part</u>:

- It has atleast one statement.

Documentation section

Link Section

Definition Section

Global Declaration Section

main() function section

{

Declaration part

Executable part

}

Subprogram section

Function1

Function2 …   user defined function

Steps in program development / Purpose of program planning

- The process of developing a computer program is to solve a specific problem involves the following steps.
  - Program definition
  - Program design
  - Program coding
  - Program compilation and execution
  - Program testing and debugging
  - Program documentation

1 . Program definition

- A precise definition of the problem that the program is required to solve is an essential prerequisite. It involves obtaining answers to the following questions.
  - What must the program do.
  - What output are required and in what form.
  - What inputs are available and in what form.

  The complete set of requirements and assumptions constitute the problem definition.

2 . Program design

- In this step a complete flow of the steps required in the development of the program is developed .
- This is usually done using either a flowchart or an algorithm. Flowchart is a pictorial representation of the steps in the program design whereas algorithm is a step by step representation of the logic to solve a problem , where each step represent a specific task to be carried out.

3 . Program coding

- Program coding refers to the instructions written in a programming language .
- We have the freedom to choose any language for coding the program.
- The user should know the clear syntax of the language .
- The process of converting the program design (with the help of algorithm and flowchart) into a computer language is called program coding .

4. Program Compilation & Execution

- Once the program has been written,it must be entered into the computer,before it is compiled and executed . This is accomplished by the use of an editor. Editors are used to create and alter text files.
- Once the program has been entered , it must be compiled and executed .The successful compilation of the program will result in the executable object program.
- After successful compilation the object program can be executed by an appropriate system command .

5. Program testing and debugging

Usually the program do not execute successfully for the first time , they contain few errors or bugs .

- There are 4 levels of debugging
- Desktop error : The programmer locates and corrects as many errors as possible by eyeing the program . The program is referred to a desk checking.
- Syntax error : Errors in the syntax of the program are usually reflected during compilation of the program and are called the compile time error / Syntax errors and creation of errors. These generate error message which are not usually straight forward , but they are helpful in identifying the nature

Program testing and debugging

Execution time errors : Certain errors can be detected only during the execution time. Such errors are called execution time errors Eg : Error to divide an arithmetic expression by zero.

- Logical errors : This arises in the design of the algorithm that cannot be detected as the syntax is correct . A buggy program produces incorrect output or it halts before producing the complete output .

6. Program Documentation

- Refers to a collection of information about the program that specifies what is being done and how .
- It includes detailed requirements of the program , the layout of all output and input data , top down decomposition , the algorithm for each component , the source testing created during the last compilation of the program , the inputs data used for testing and the output of this test run
- Also includes a users guide that provides procedural instructions for installing and executing the program

Algorithm

- The term algorithm may be defined as a sequence of instruction, designed in such a way that if the instructions are executed in a specific sequence , the desired result will be obtained .
- The algorithm represents the logic of the processing to be performed .

Characteristics of an algorithm

Algorithm should posses the following characteristics

- Each and every instruction should be precise and unambiguous .
- Each instruction should be such that it can be performed in a finite time .
- One or more instructions should not be repeated infinitely .This ensures algorithm will terminate.
- After the algorithm terminates, the desired result must be obtained.

FLOWCHART

- A flowchart is a pictorial representation of an algorithm that uses boxes of different shapes to denote , different types of instructions .
- The actual instructions are written within these boxes using clear and concise statements .
- The boxes are connected by solid lines having arrow marks to indicates the flow of the operation are to be executed .
- A flowchart is a picture of the logic to be included in the computer program .

FLOWCHART

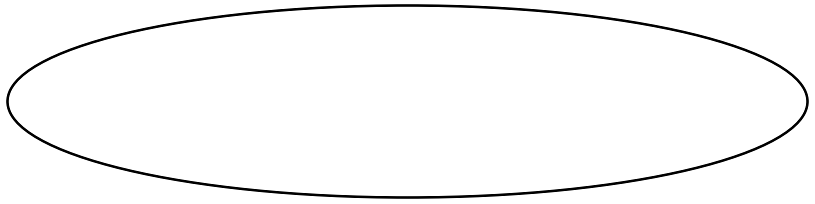
- It is simply a method of assisting the programmer to layout in a visual two dimensional format ideas of how to organize a sequence of steps necessary to solve a problem by a computer.
- The main advantage of the flowchart is that while drawing a flowchart, one is not concerned with the details of the element of the programming language. Hence can fully concentrate on the logic of the procedure .
- Since the flowchart shows the flow of the operation in pictorial form, any error in logic can be detected more easily.

FLOWCHART SYMBOLS

- Flowchart uses boxes of different shapes to denote different types of instructions . The symbols and their function are discussed below

- Start/end

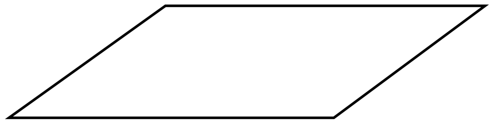
This symbol is used to indicate the starting point and end point of the program logic flow.



FLOWCHART SYMBOLS

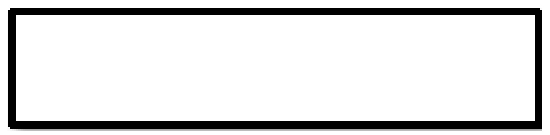
- **INPUT / OUTPUT**

Input output symbols are used to denote any input/output function used in the program.



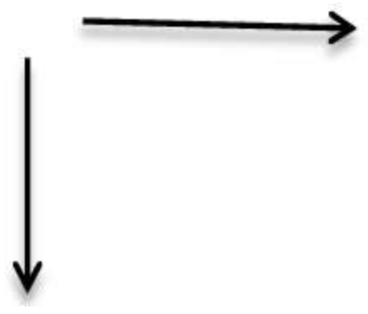
FLOWCHART SYMBOLS

- **PROCESSING**
- The processing symbol is used in the Flowchart to represent arithmetic and data movement instructions.
- When more than one arithmetic and data movement instructions are to be executed they are normally placed in the same processing box and are assumed to be executed in the order of their appearance



FLOWCHART SYMBOLS

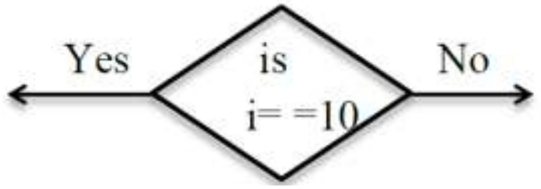
- **FLOW LINES**
- Flow lines are arrow lines that are used to indicate the flow of action ie, the exact sequence in which the instructions are to be executed .
- The normal flow of flowcharts is from top to bottom and left to right .
- Flow lines should not cross each other and such instructions should be avoided.



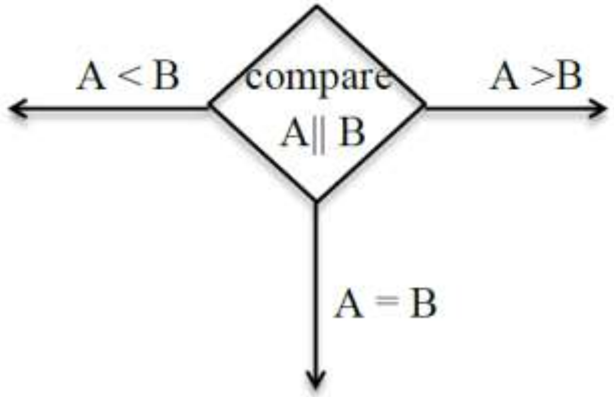
DECISION

- The decision symbol is used in the flowchart to indicate a point at which the decision is to be made and a branch to one or more alternate points are possible . There are 3 different ways of branching:

1 . Two way branch

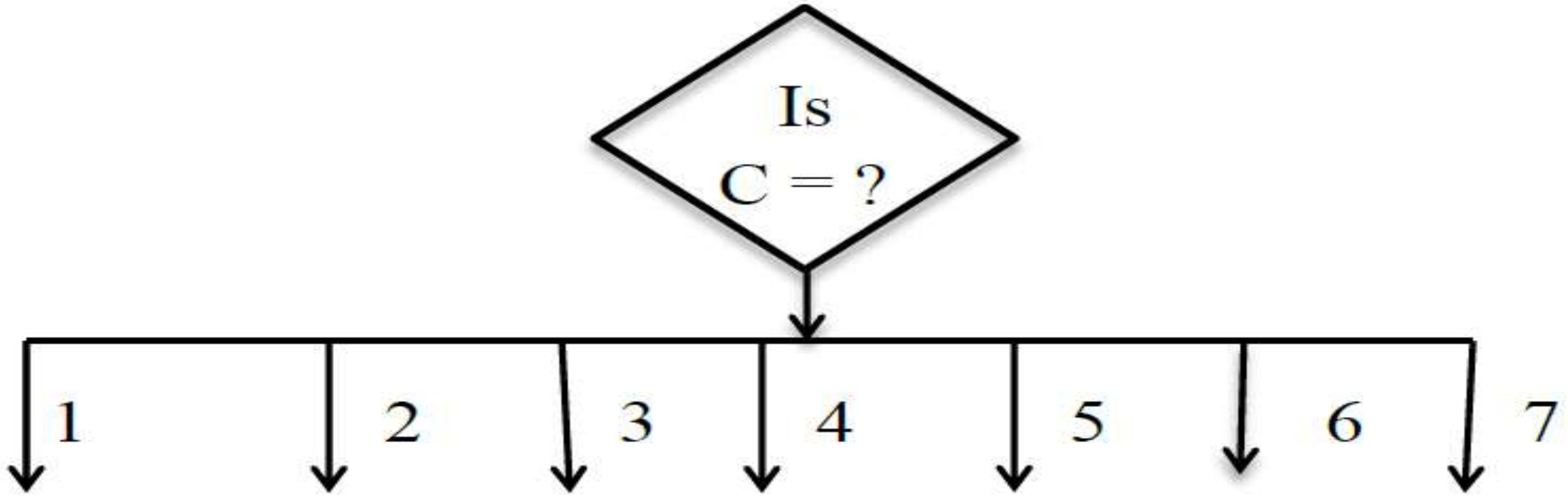


2 . Three way branch



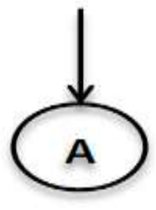
DECISION

3 . Multiple way branch



CONNECTORS

- If a flowchart is too long , the flow lines start criss- crossing at many points and this cause confusion and reduces understandability of the flowcharts .
- There are instances when flowchart become too long to fit in a single page and use of flow lines becomes impossible . In such cases it is useful to utilize the connector symbols as a substitute for flow lines
- The connector symbol represented by a circle and a letter or digit is placed in the circle to indicate the link .A pair of identically labelled , connector symbols used to indicate a continuous flow , when the use of the lines are confusing . So two connectors with identical labels serve the same function as a long flow line . Connector do not represent any operation and there use in the flowchart is only for the sake of convenience.



PROGRAMS

- Programs are usually either one or combination of the following
  - Sequence
  - Selection
  - Iteration

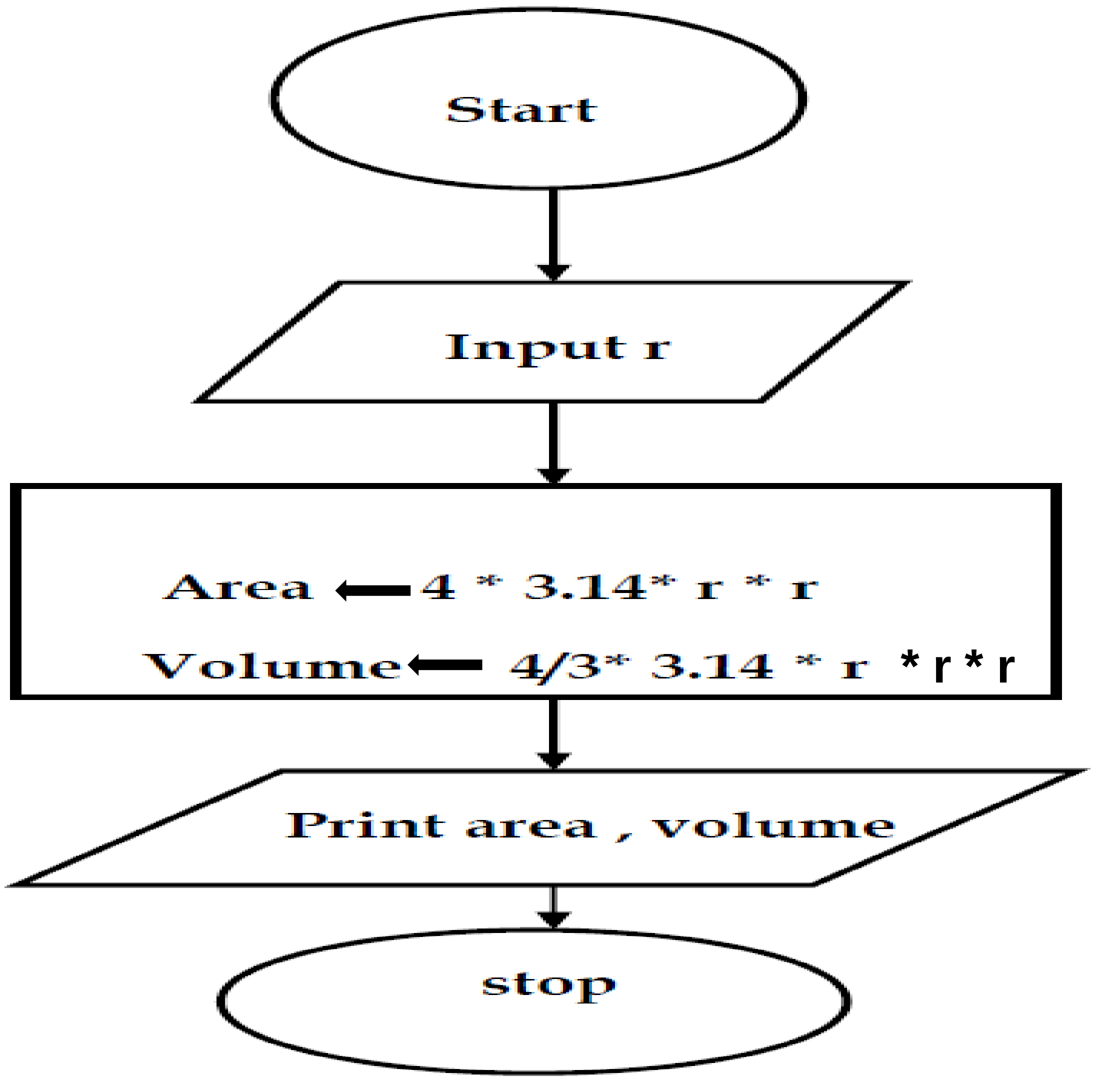
SEQUENCE

Q: Find the area and volume of a SPHERE

ALGORITHM

```
1 : Start
2 : input r
3 : area ⟵ 4 *3.14 * r * r
4 : volume ⟵ 4/3 *3.14 * r * r*r
5 : print area , volume
6 : Stop
```

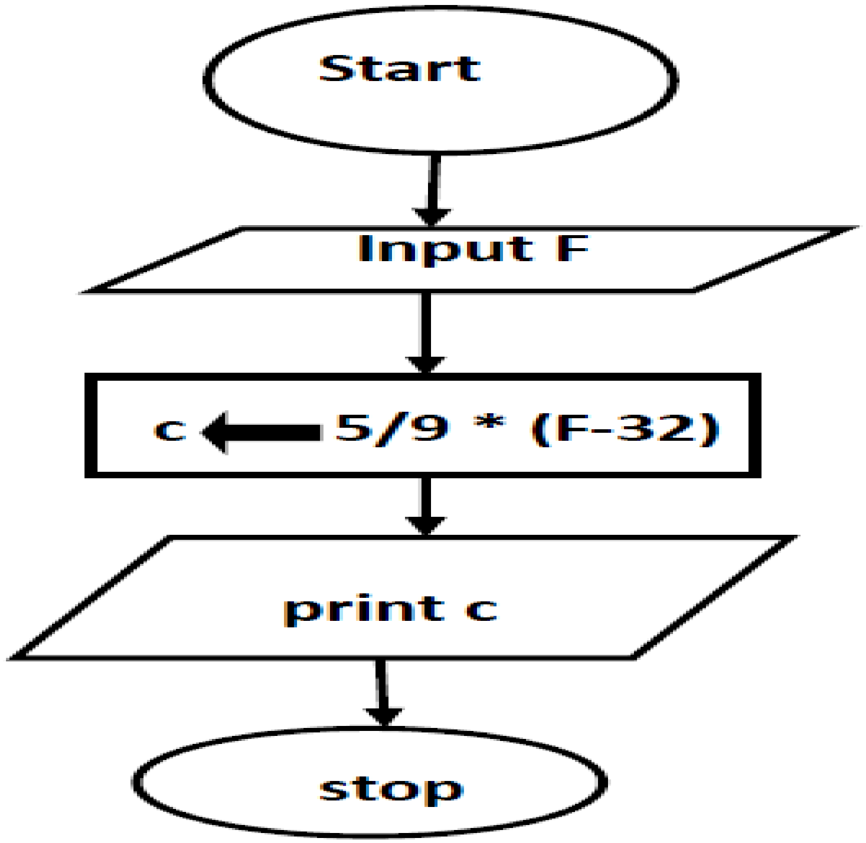
FLOWCHART


Start
Input r
Area ⟵ 4 * 3.14* r * r
Volume⟵ 4/3* 3.14 * r *r*r
Print area , volume
stop

*Q : Program to convert farenheit to Celsius [c = 5/9 *(F -32)]*

- **ALGORITHM**
- **1 : Start**
- **2 : input F**
- **3 : c ⟵ 5/9 * (F-32)**
- **4 : print c**
- **5 : Stop**

FLOWCHART



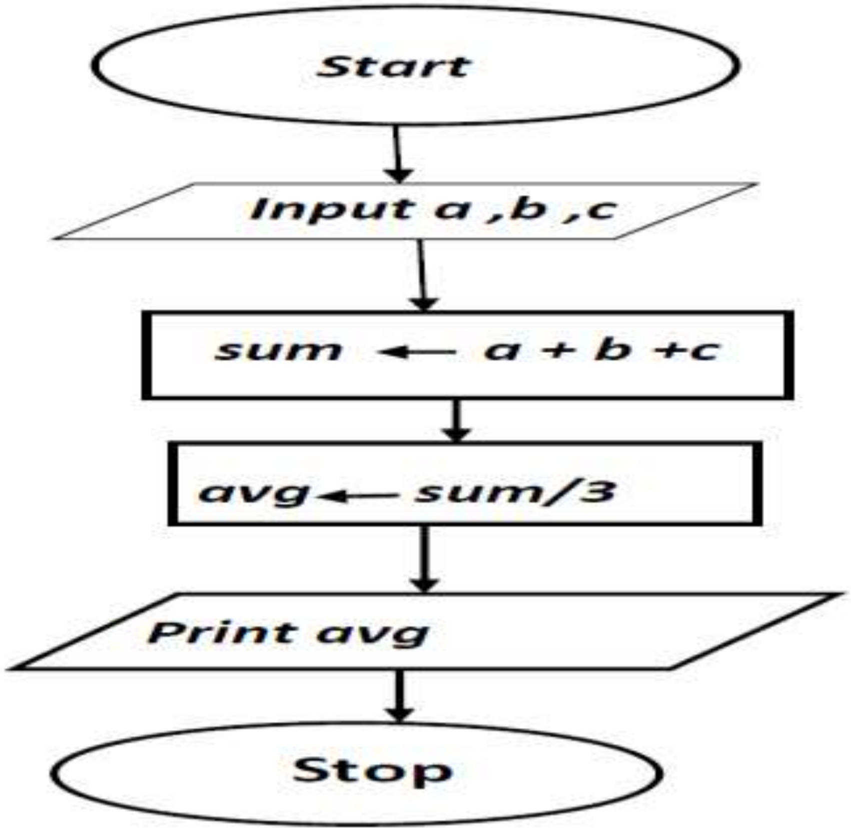
Q. Program to add 3 numbers and find its Average

ALGORITHM

```
1 : Start
2 : input a , b , c
3 : sum ← a+b+c
4 : avg ← sum/3
5 : Print avg
6 : Stop
```

FLOWCHART

- flowchart

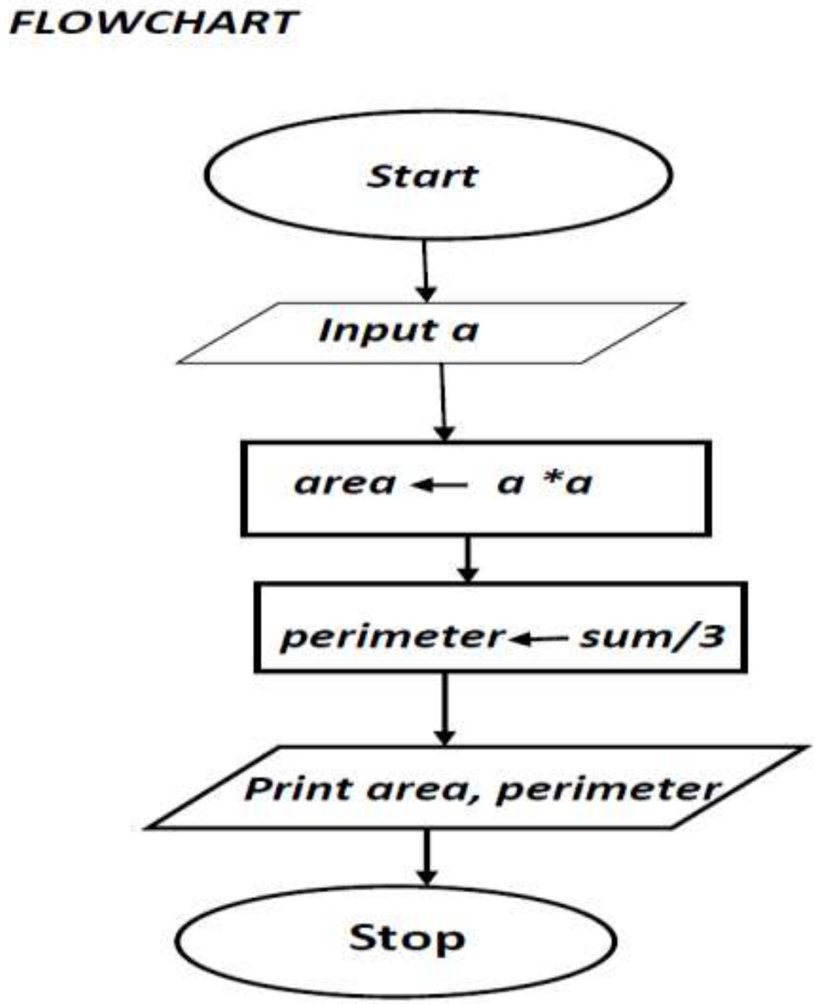


Q: Program to find the area and perimeter of a square

ALGORITHM

```
1 : Start
2 : input a
3 : area $\leftarrow$ a*a
4 : perimeter $\leftarrow$ 4*side
5 : Print area,perimeter
6  : Stop
```

FLOWCHART

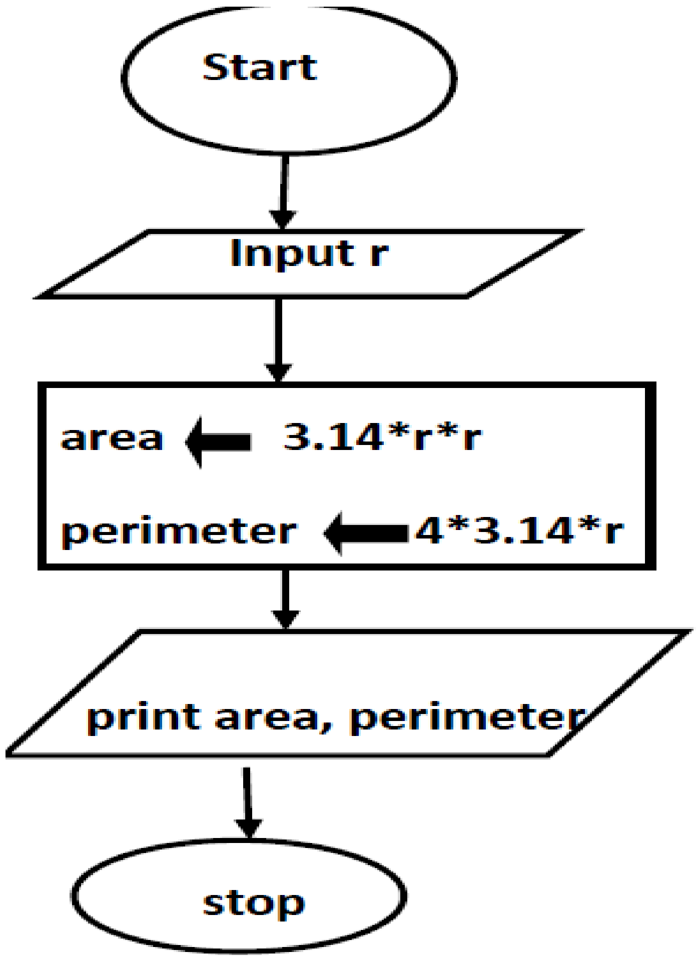

FLOWCHART
Start
Input a
area ← a *a
perimeter ← sum/3
Print area, perimeter
Stop

Program to find the area and perimeter of a circle

ALGORITHM

```
1 : Start
2 : input r
3 : area =3.14 *r*r
    perimeter= 4*3.14*r
4 : print area, perimeter
5 : Stop
```

FLOWCHART

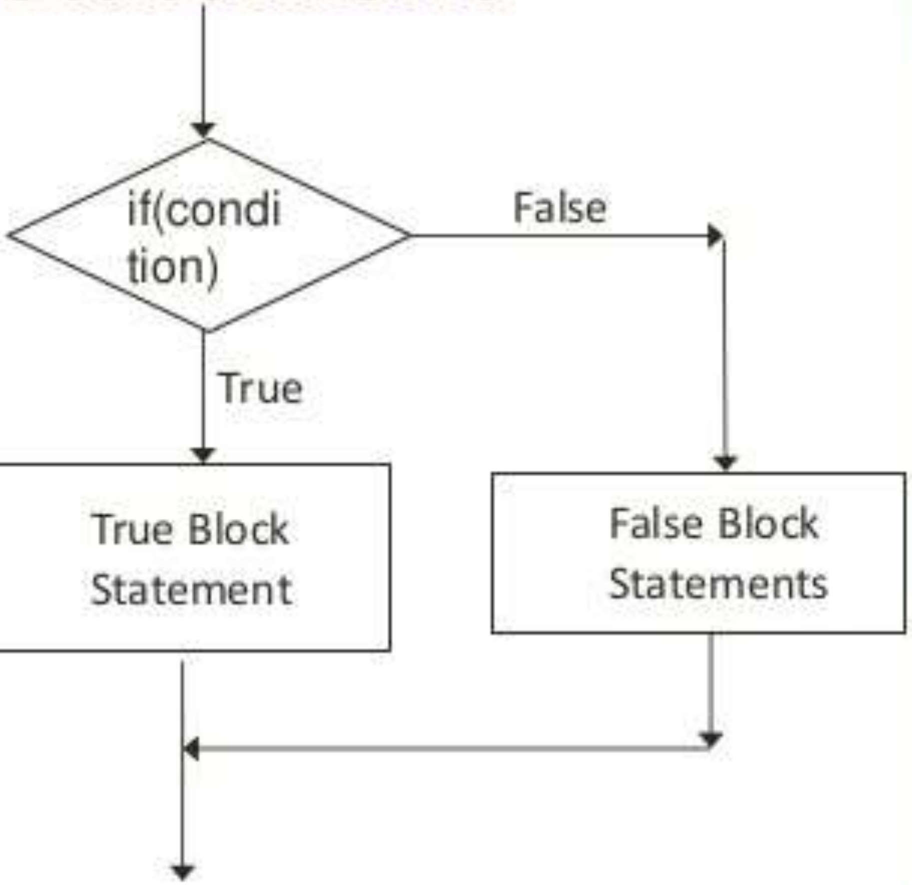

Start
Input r
area ⟵ 3.14*r*r
perimeter ⟵ 4*3.14*r
print area, perimeter
stop

SELECTION

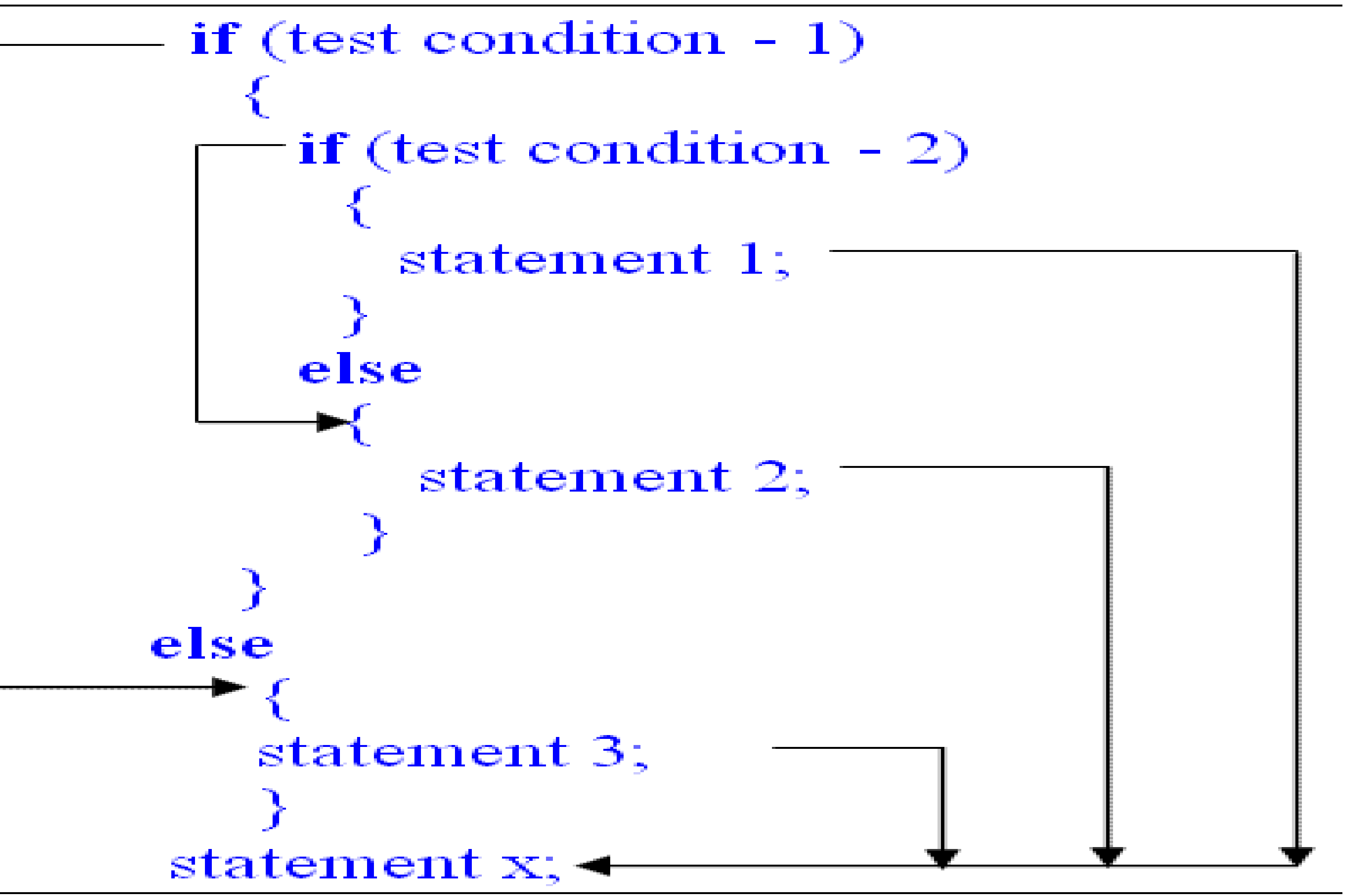
if - else statement

Syntax:

```
if(condition)
{
True block statements;
}
else
{
False block statements;
}
```

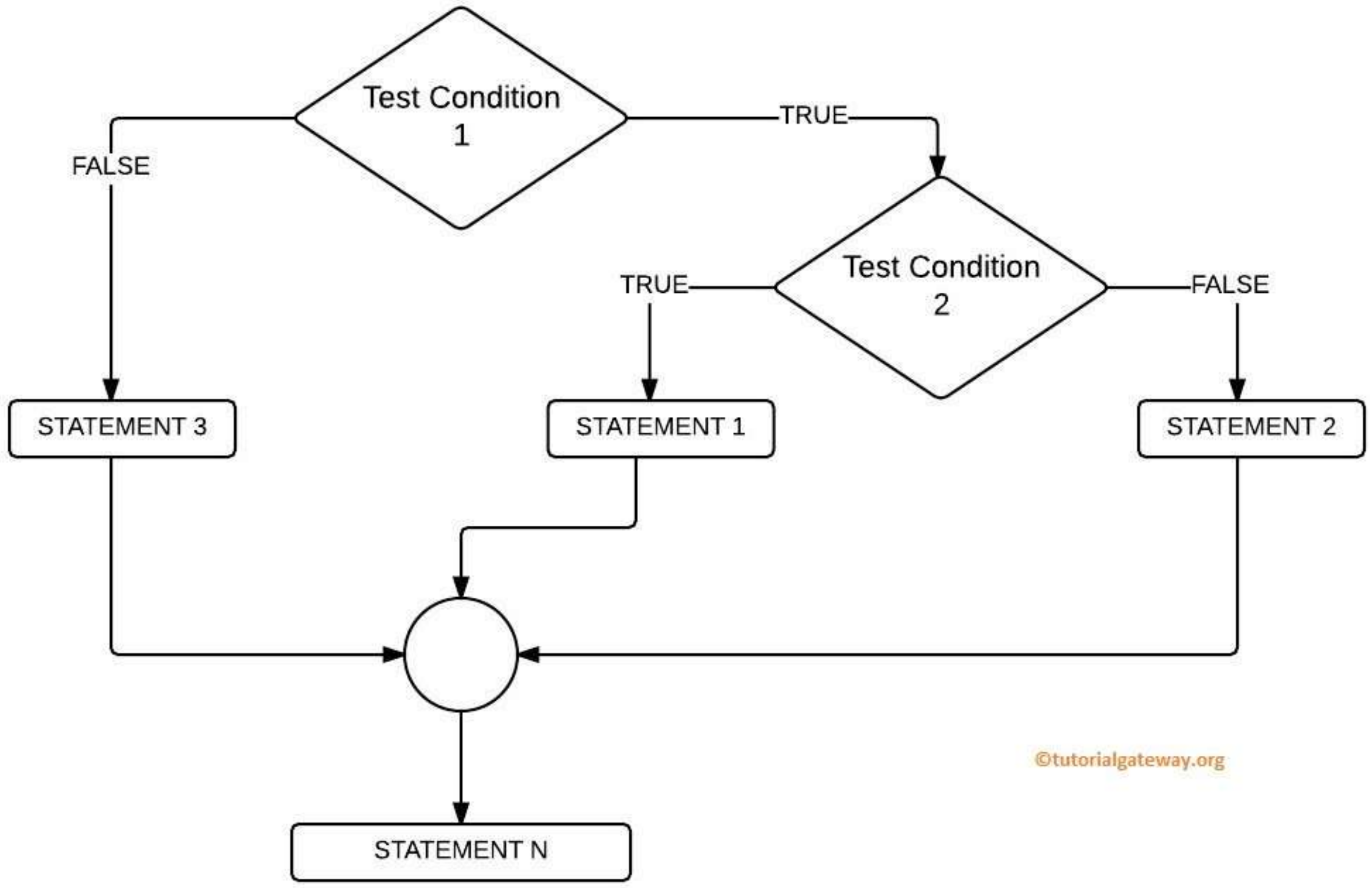


NESTED IF STATEMENT



```
if (test condition - 1)
  {
    if (test condition - 2)
      {
        statement 1;
      }
    else
      {
        statement 2;
      }
  }
else
  {
  statement 3;
  }
statement x;
```

NESTED IF FLOWCHART

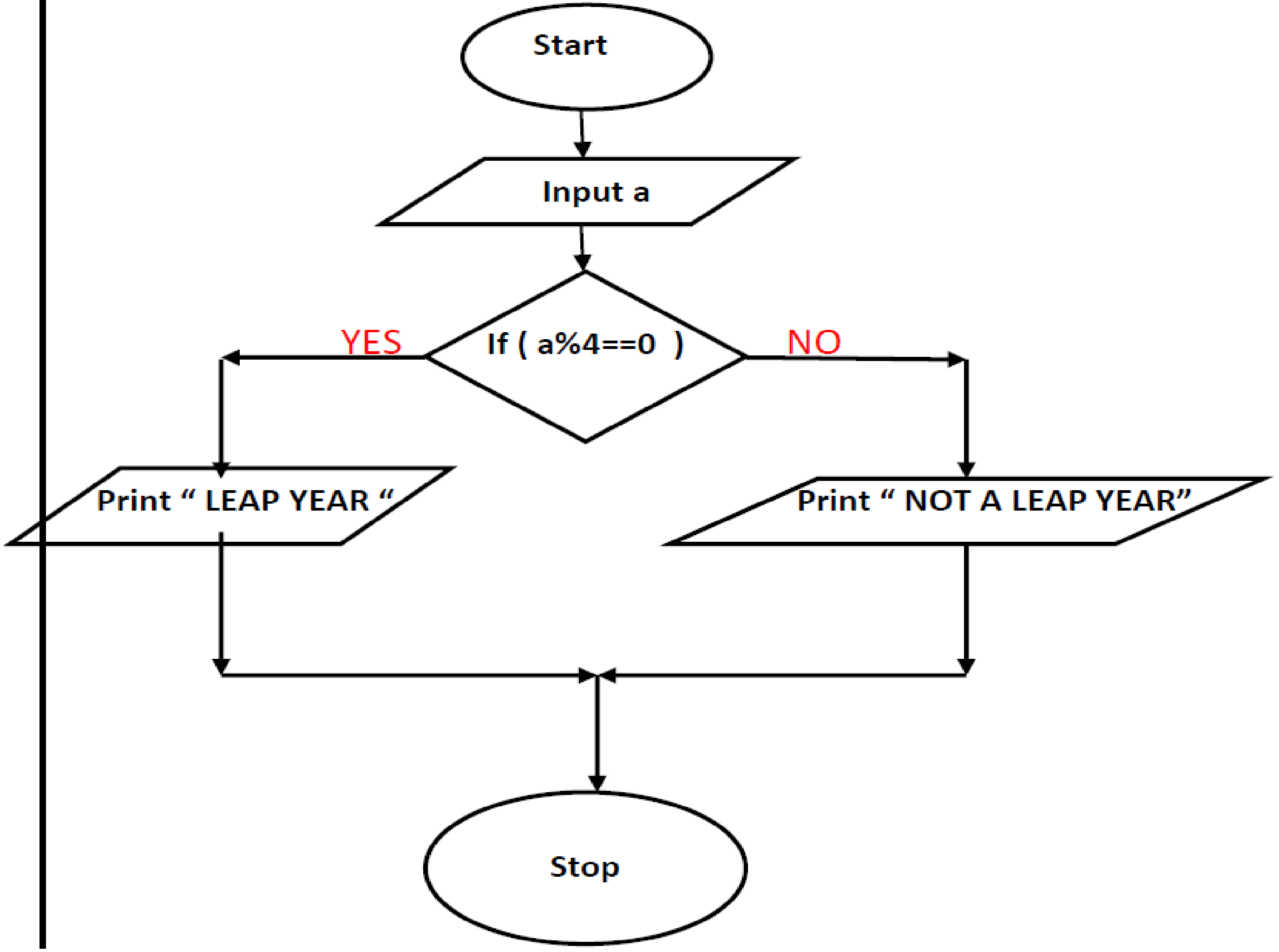


Q : Check whether a year is Leap year or not

- **ALGORITHM**

```
1 : Start
2 : input a
3 : if ( a%4==0 )
        Print "LEAP YEAR"
    else
        print " NOT A LEAP YEAR"
4 : Stop
```

FLOWCHART

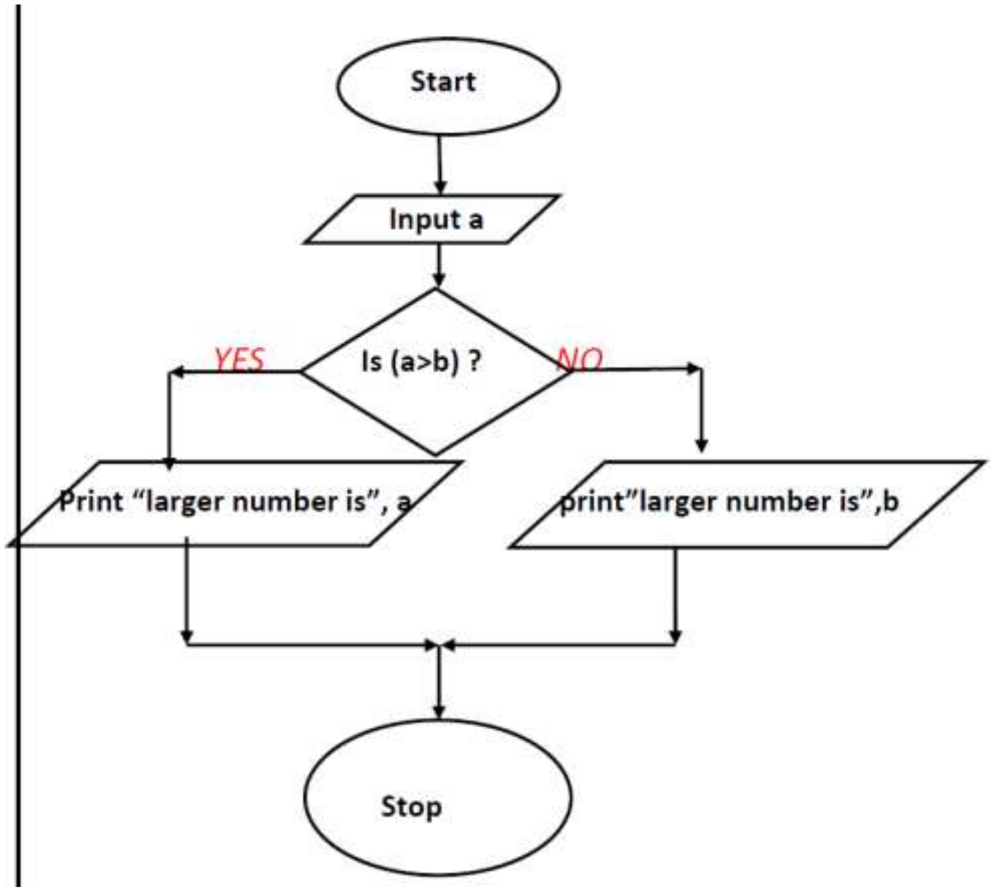

Start
Input a
If (a%4==0)
YES
NO
Print " LEAP YEAR "
Print " NOT A LEAP YEAR"
Stop

Q: PROGRAM TO FIND THE LARGER OF TWO NUMBERS

ALGORITHM

```
1 : Start
2 : input a,b
3 : if ( a > b)
      print "larger number is", a
else
      print "larger number is" b
4 : Stop
```

FLOWCHART

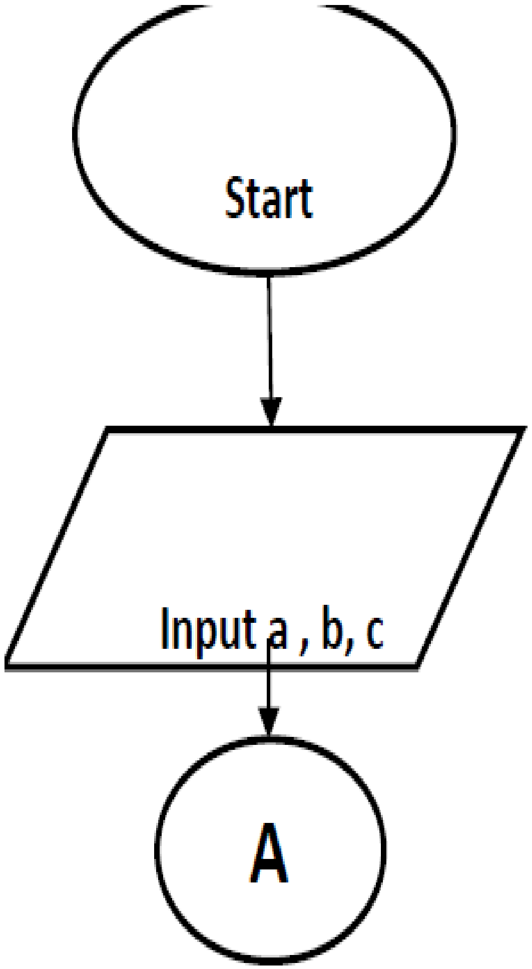

Start
Input a
Is (a>b) ?
YES
NO
Print “larger number is”, a
print”larger number is”,b
Stop

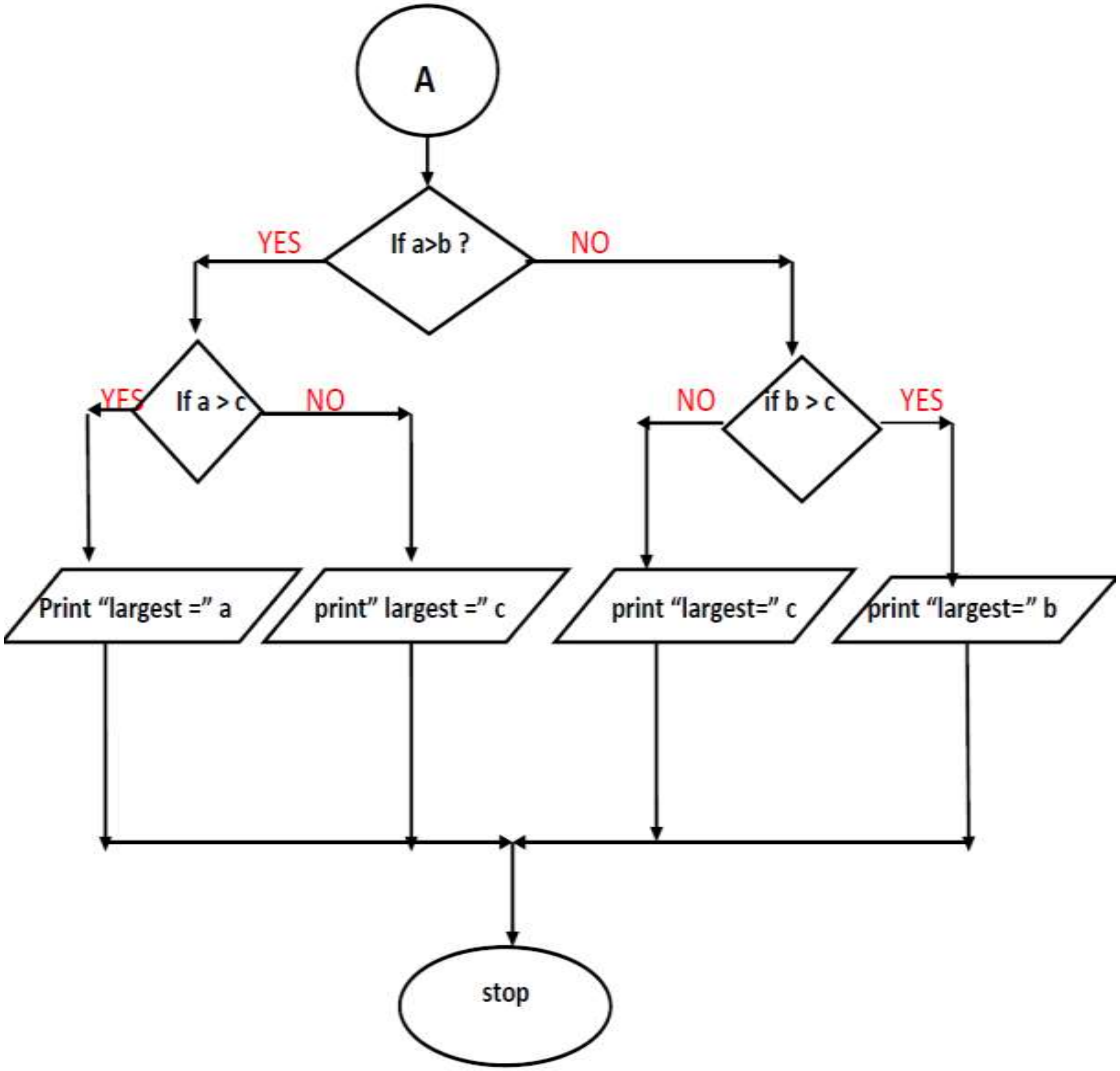
Q : Find largest among 3 numbers

ALGORITHM

```
1 : Start
2 : input a , b , c
3 : if (a>b)
     {
       If (a>c)
          print “ largest = ” a
      else
         print “ largest = “ c
      }
   else
     {
        If (b>c)
           print “ largest = “ b
   else
          print “largest =” c
     }
4 : Stop
```

FLOWCHART

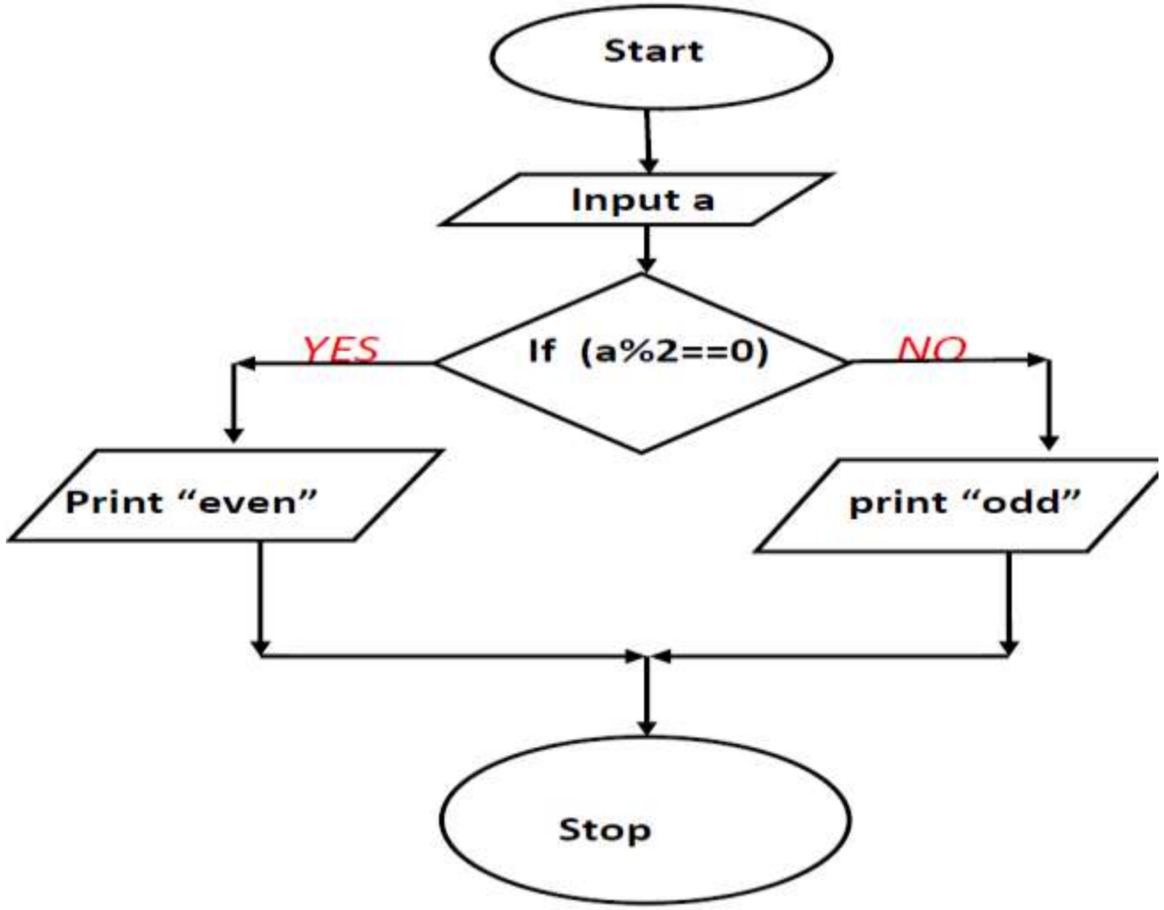

Start
Input a , b, c
A


A
If a>b ?
YES
NO
If a > c
YES
NO
if b > c
NO
YES
Print "largest =" a
print" largest =" c
print "largest=" c
print "largest=" b
stop

Q : Program to accept a number and check whether it’s even or odd

```
1 : Start
2 : input a
3 : if ( a % 2==0)
      print “even”
      else
      print “odd”
4 : Stop
```

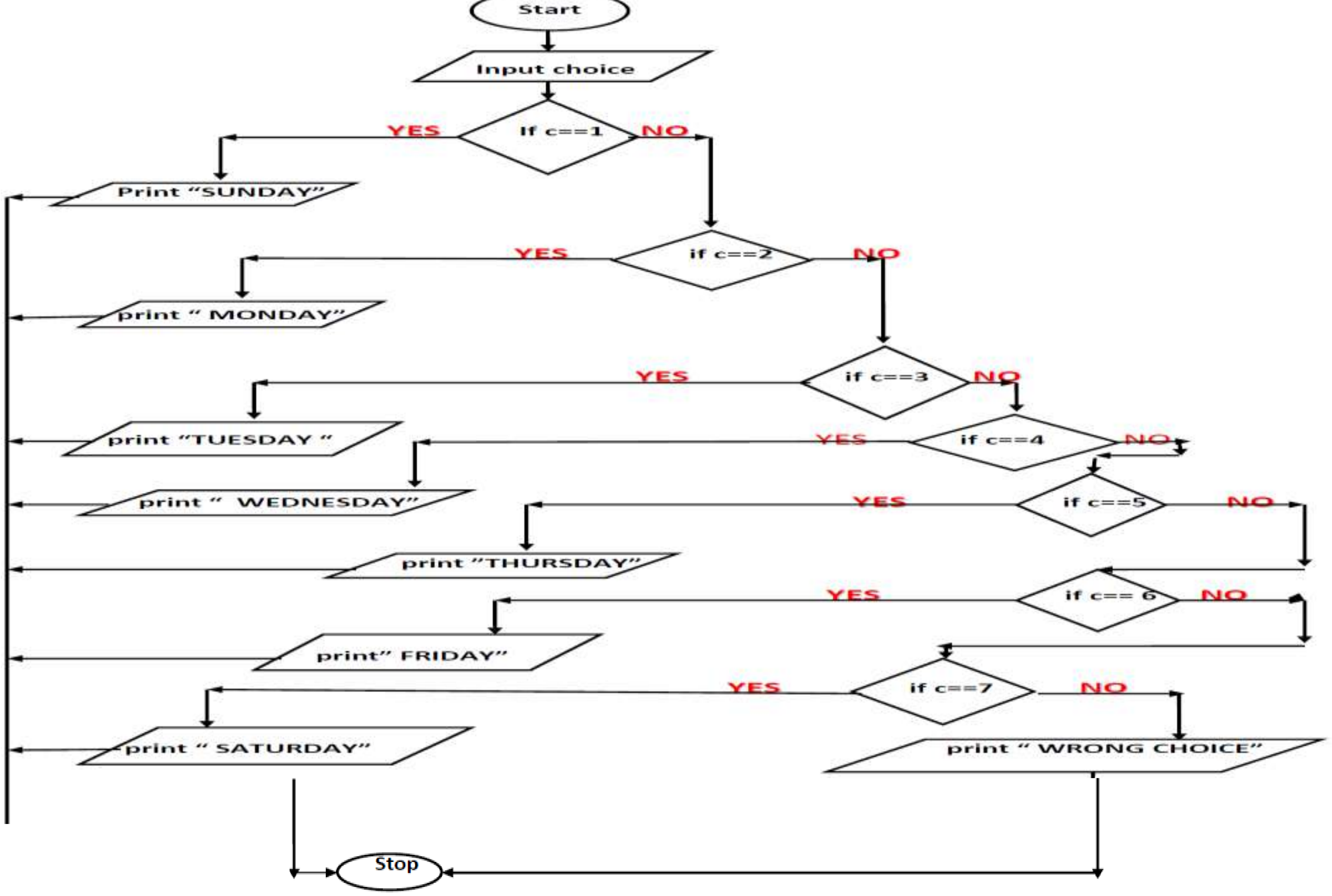
FLOWCHART



Q : Enter a number from 1 to 7. If 1 print SUNDAY , 2 print MONDAY ....else print WRONG CHOICE

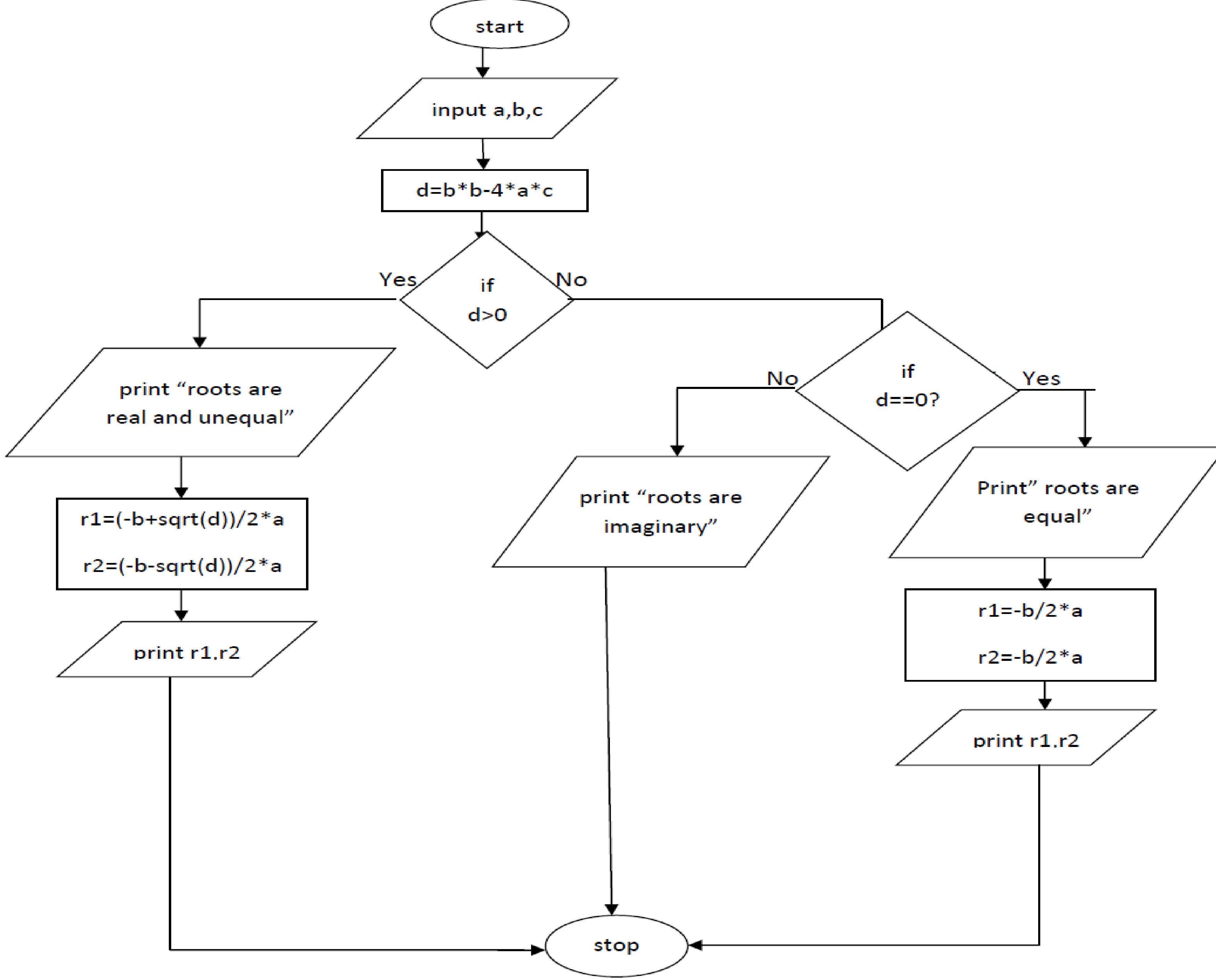
- **ALGORITHM**

```
1 : Start
2 : input
3 : if ( c == 1 )
     print "SUNDAY"
     else if (c ==2)
         print "MONDAY"
         else if (c== 3)
              print "TUESDAY"
              else if (c==3)
                  print" WEDNESDAY"
                  else if(c==4)
                       print "THURSDAY"
                       else if (c==5)
                            print "FRIDAY"
                            else if (c==6)
                                 print "FRIDAY"
                                 else if (c==7)
                                      print "SATURDAY"
                                      else
                                      print "WRONG CHOICE"
4 : Stop
```


Start
Input choice
If c==1
YES
NO
Print "SUNDAY"
if c==2
YES
NO
print " MONDAY"
if c==3
YES
NO
print "TUESDAY "
if c==4
YES
NO
print " WEDNESDAY"
if c==5
YES
NO
print "THURSDAY"
if c== 6
YES
NO
print" FRIDAY"
if c==7
YES
NO
print " SATURDAY"
print " WRONG CHOICE"
Stop

```
1 : start
2 : input a ,b ,c
3 : d=b*b - 4*a*c
4 : if (d>0)    {
        print “Roots are real and unequal”
        r=(- b + sqrt (d))/(2*a)
        r2=(- b -  sqrt (d))/(2*a)
        print r1,r2
        }
else if (d==0)
        {
        print “roots are equal”
        r1=-b/2*a
        r2=-b/2*a
        print r1,r2
        }
        else
        print “roots are imaginary”

5.Stop
```


start
input a,b,c
d=b*b-4*a*c
if d>0
Yes
No
print "roots are real and unequal"
r1=(-b+sqrt(d))/2*a
r2=(-b-sqrt(d))/2*a
print r1,r2
if d==0?
No
Yes
print "roots are imaginary"
Print" roots are equal"
r1=-b/2*a
r2=-b/2*a
print r1,r2
stop

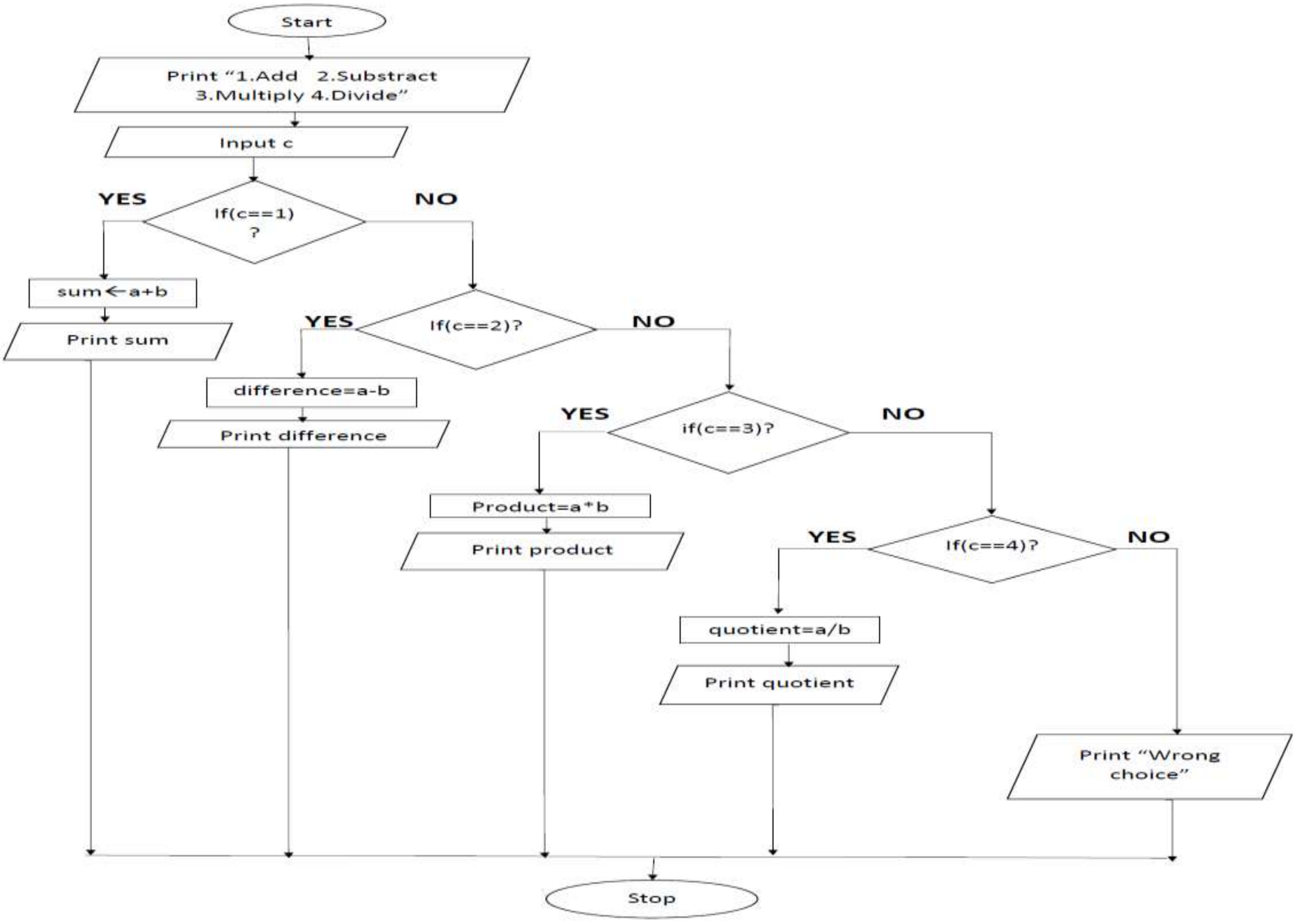
Program to add, subtract, multiply two numbers using switch statement.

ALGORITHM

```
    1.start
    2.print " 1.add, 2. substract 3. multi 4.divide"
    3.input c
    5. if(c==1)
    sum ← a+b
    Print sum
    else if (c==2)
    Difference ← a-b
Print "difference"
        else if (c==3)
        Product ← a*b
            Print "product"
            else if(c==4)
                Quotient ← a/b
            Print "Quotient"
        else
            Print "wrong choice"

        6.stop
```

FLOWCHART:


Start
Print "1.Add 2.Substract 3.Multiply 4.Divide"
Input c
YES
If(c==1) ?
NO
sum←a+b
Print sum
YES
If(c==2)?
NO
difference=a-b
Print difference
YES
if(c==3)?
NO
Product=a*b
Print product
YES
If(c==4)?
NO
quotient=a/b
Print quotient
Print "Wrong choice"
Stop

ITERATION

LOOPS IN C

C programming has three types of loops:

- for loop
- while loop
- do...while loop

FOR LOOP-SYNTAX

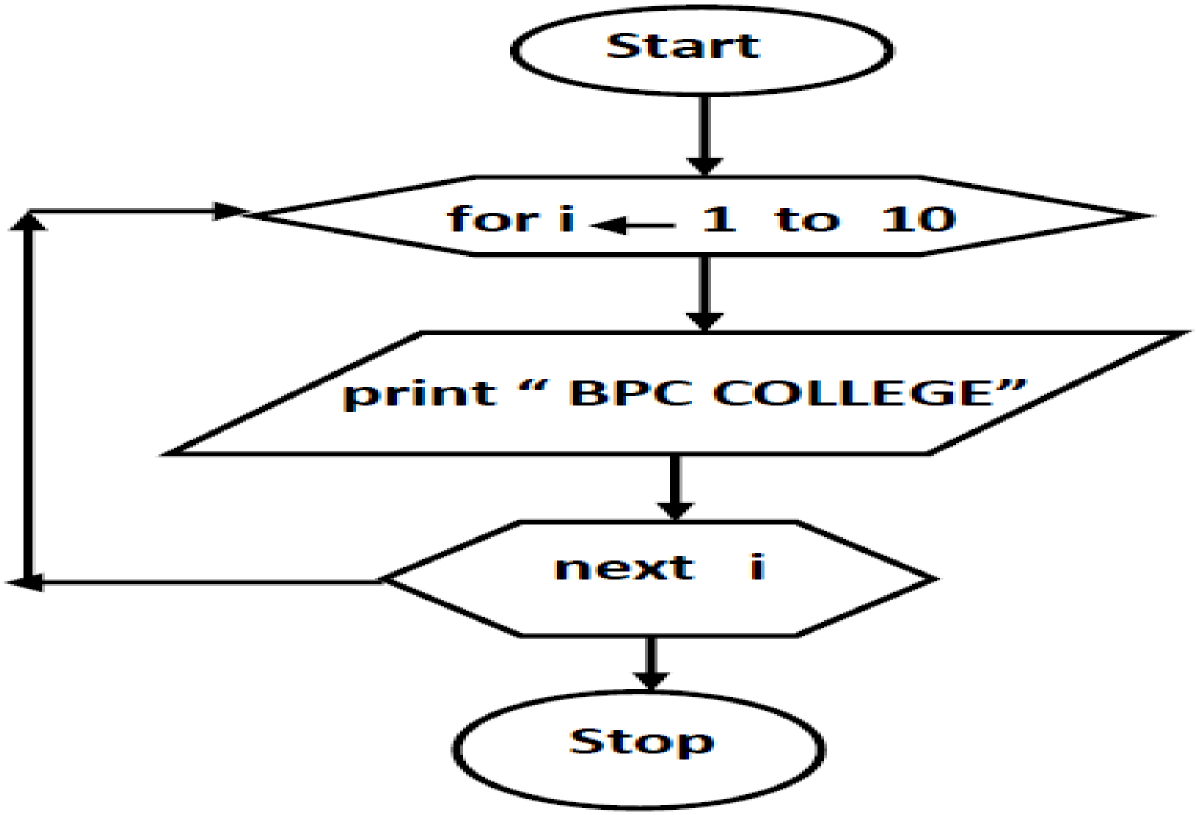
```
for ( initialization; condition;
increment/decrement )

{
statement(s);
}
```

Q : Program to print BPC COLLEGE 10 times

```
1 : Start
2. for i ← 1 to 10
   print ( “BPC COLLEGE”)
3. Stop
```

FLOWCHART

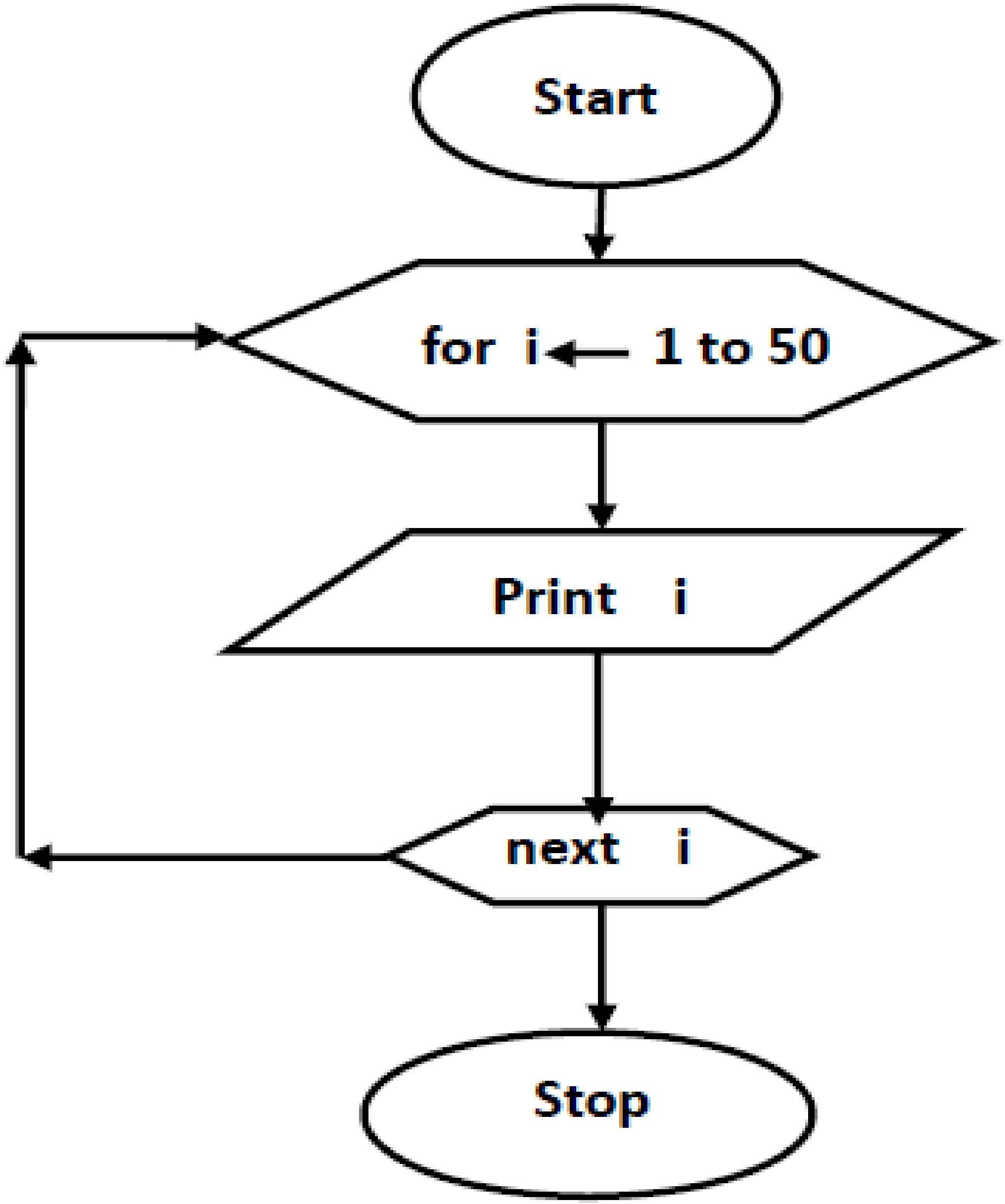


Q : Program to print whole numbers between 1 and 50

ALGORITHM

```
1 : Start
2 : for i $\leftarrow$ 1 to 50
    Print    i
3 : Stop
```

FLOWCHART

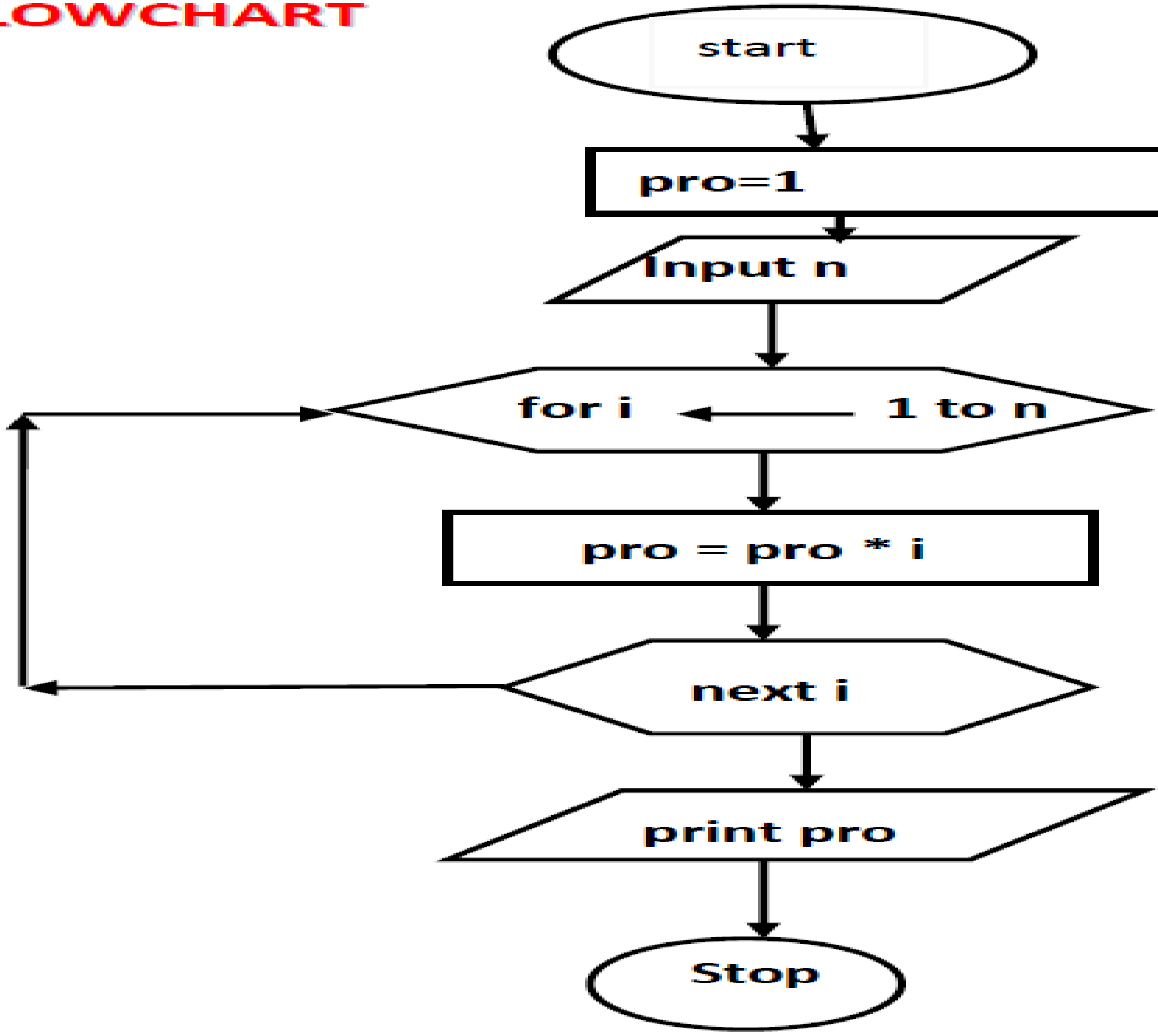


Q : Product of n natural numbers

ALGORITHM

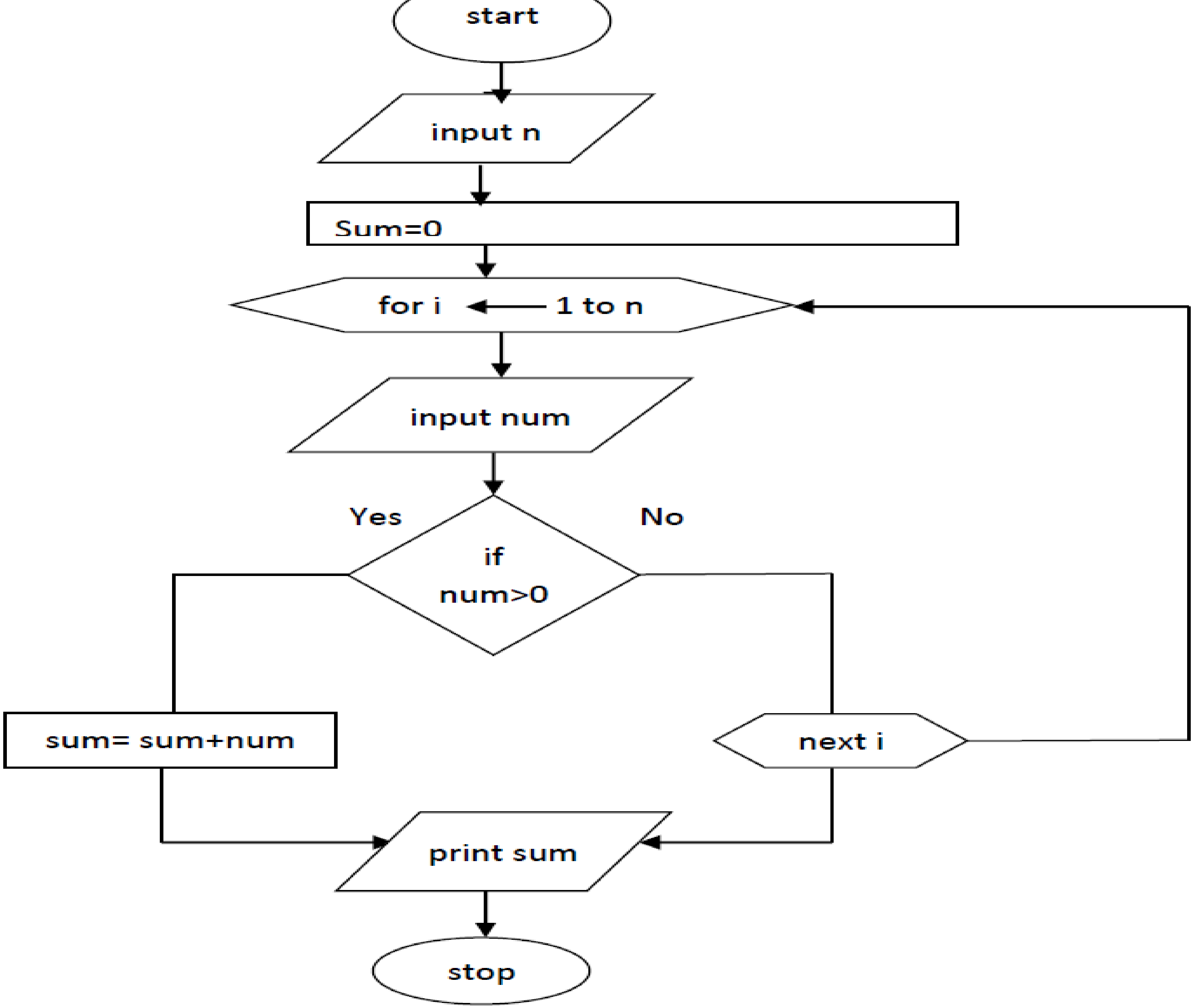
```
1 : Start
2 : input n
3 : pro=1
4:for  i ← 1 to n
     pro = pro * i
5 : print pro
6 : Stop
```

FLOWCHART



Write a program to input numbers and find the sum of positive natural numbers

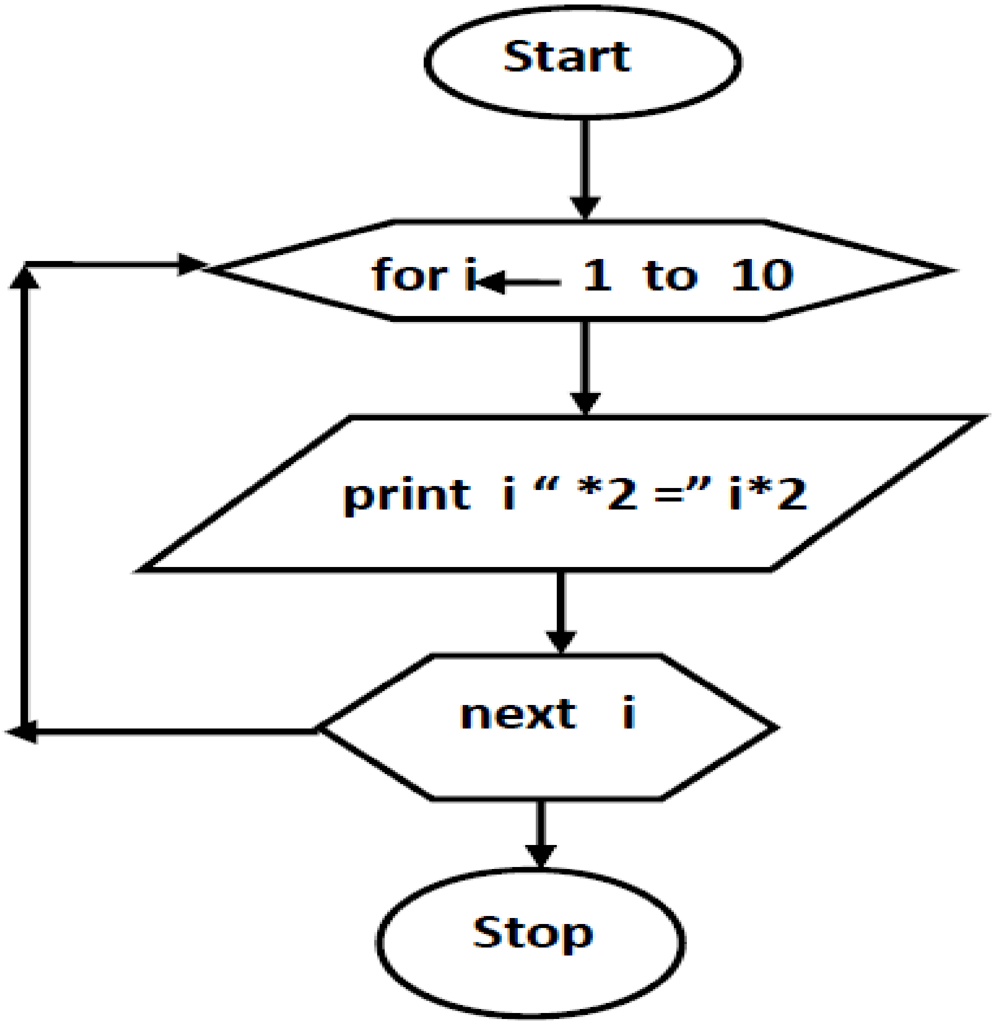
```
• 1 : start
  2 : sum=0
  3 : input n
  4 : for i ⟵ 1 to n
        input num
        if (num>0)
        sum=sum+num
        endfor
5 : print sum
6 : stop
```


start
input n
Sum=0
for i ← 1 to n
input num
Yes
No
if num>0
sum= sum+num
next i
print sum
stop

Q : Program to print multiplication table of 2

- ALGORITHM
- 1 : Start
- 2 : for i $\longleftarrow$ 1 to 10
-     Print i * 2
- 3 : Stop

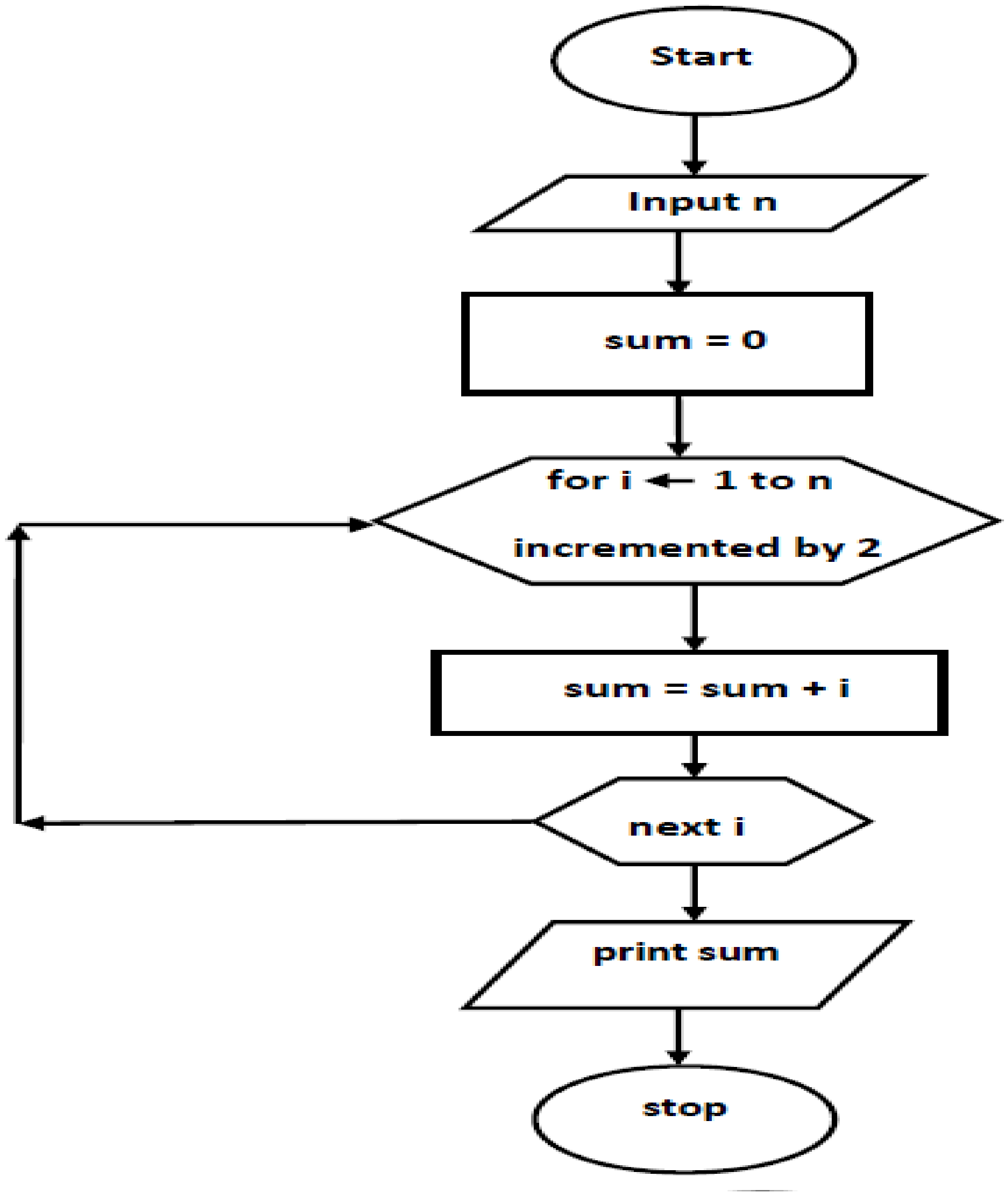
FLOWCHART



Q : Find the sum of odd numbers upto n

ALGORITHM

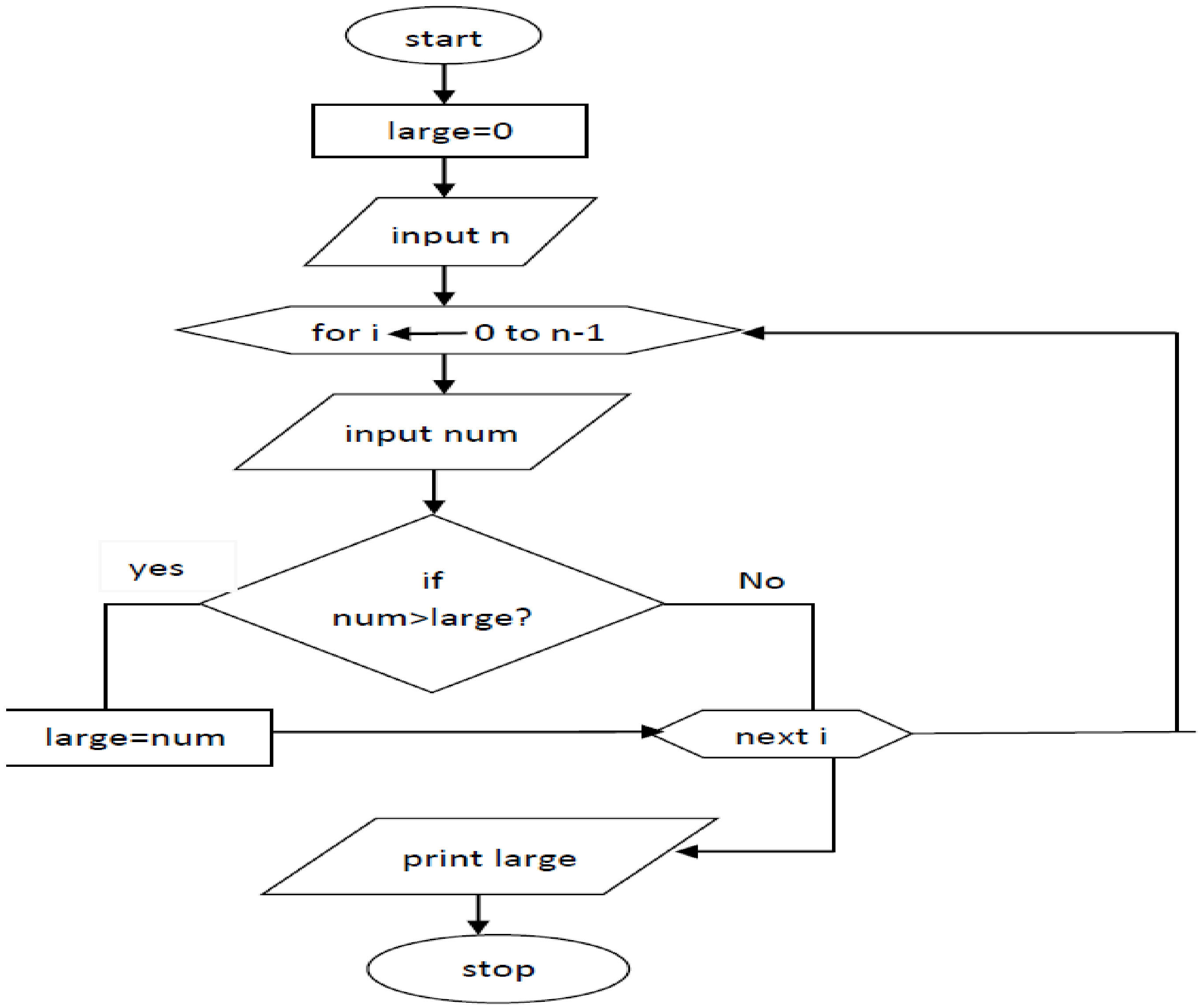
```
1 : Start
2 : input n
3 : sum  = 0
4 : for  i ← 1  to n    incremented by 2
            sum = sum + i
          next i
5 : print sum
6 : Stop
```


Start
Input n
sum = 0
for i ← 1 to n
incremented by 2
sum = sum + i
next i
print sum
stop

Find the largest of n numbers

```
1 : start
2 : large=0
3 : for i $\leftarrow$ 0 to n-1
        print "enter the number"
        input num
if num>large
        large=num
    next i
4 : print large
5 : stop
```

FLOWCHART

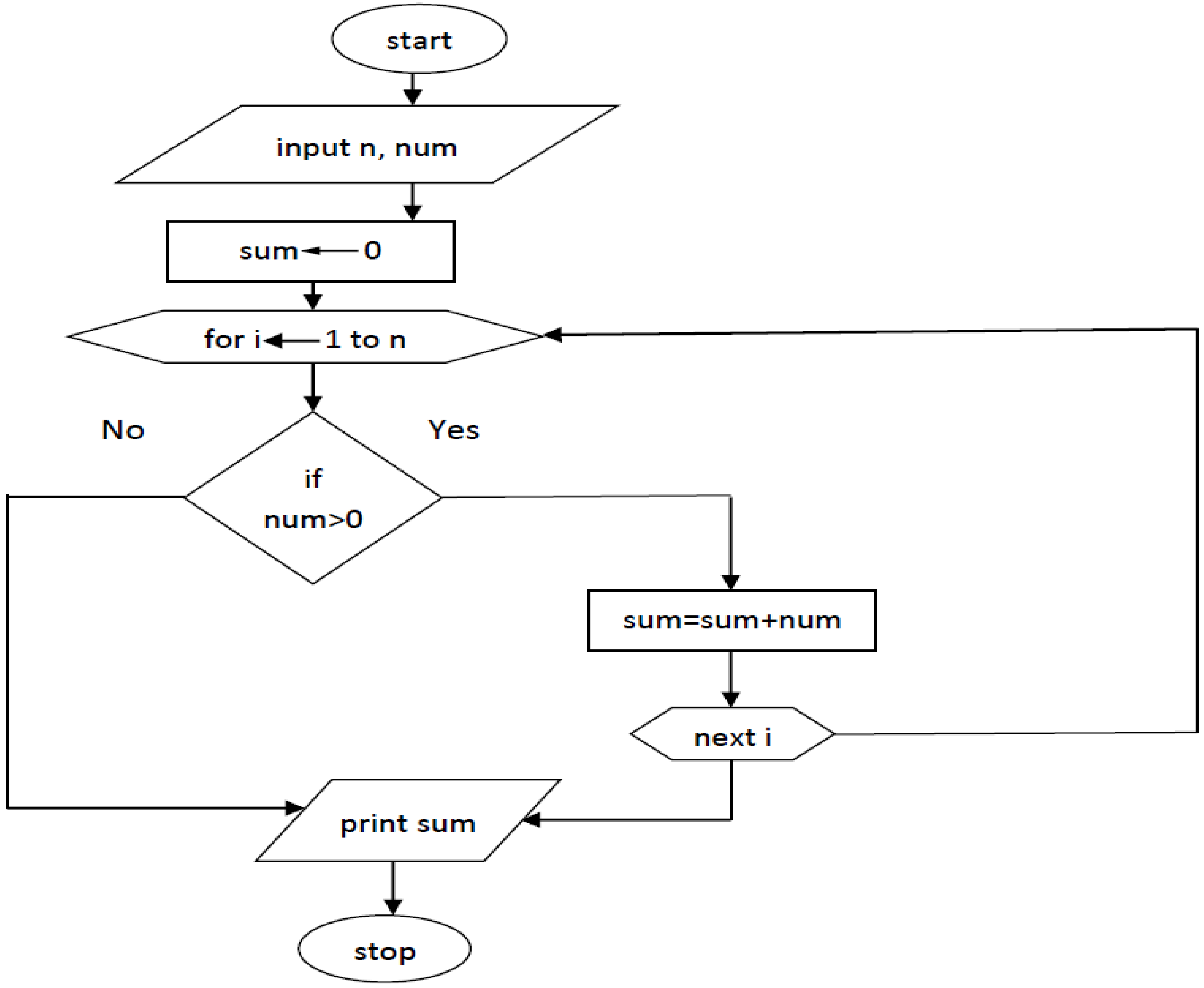

start
large=0
input n
for i ← 0 to n-1
input num
if num>large?
yes
No
large=num
next i
print large
stop

Write a program to input numbers and find its sum and come out of loop if negative number is entered.

ALGORITHM

```
1 : start
2 : input n, num
3: sum   ← 0
4 : for i   ← 0 to n
    if (num>0)
            sum=sum+num
      else
            go to 5
   next i
5 : print sum
6 : stop
```

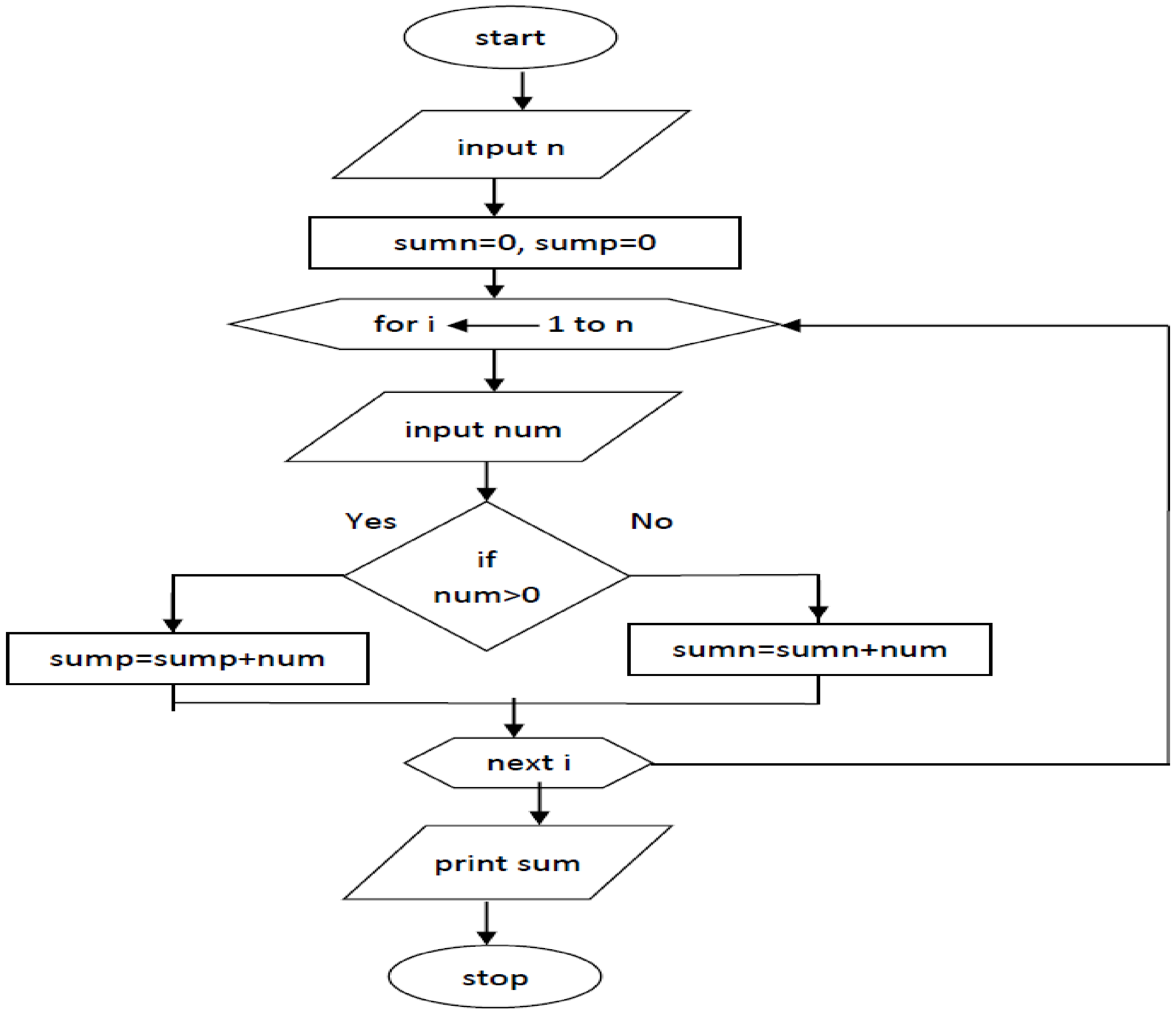
FLOWCHART


start
input n, num
sum ← 0
for i ← 1 to n
No
Yes
if num>0
sum=sum+num
next i
print sum
stop

Write a program to input numbers and find the sum of positive numbers and negative numbers separately

ALGORITHM

```
1 : start
2 : sumn=0, sump=0
3 : input n
4 : for i ← 1 to n
        input num
        if (num>0)
            sump=sump+num
        else
            sumn=sumn+num
    next i
6 : print sump, sumn
7 : stop
```

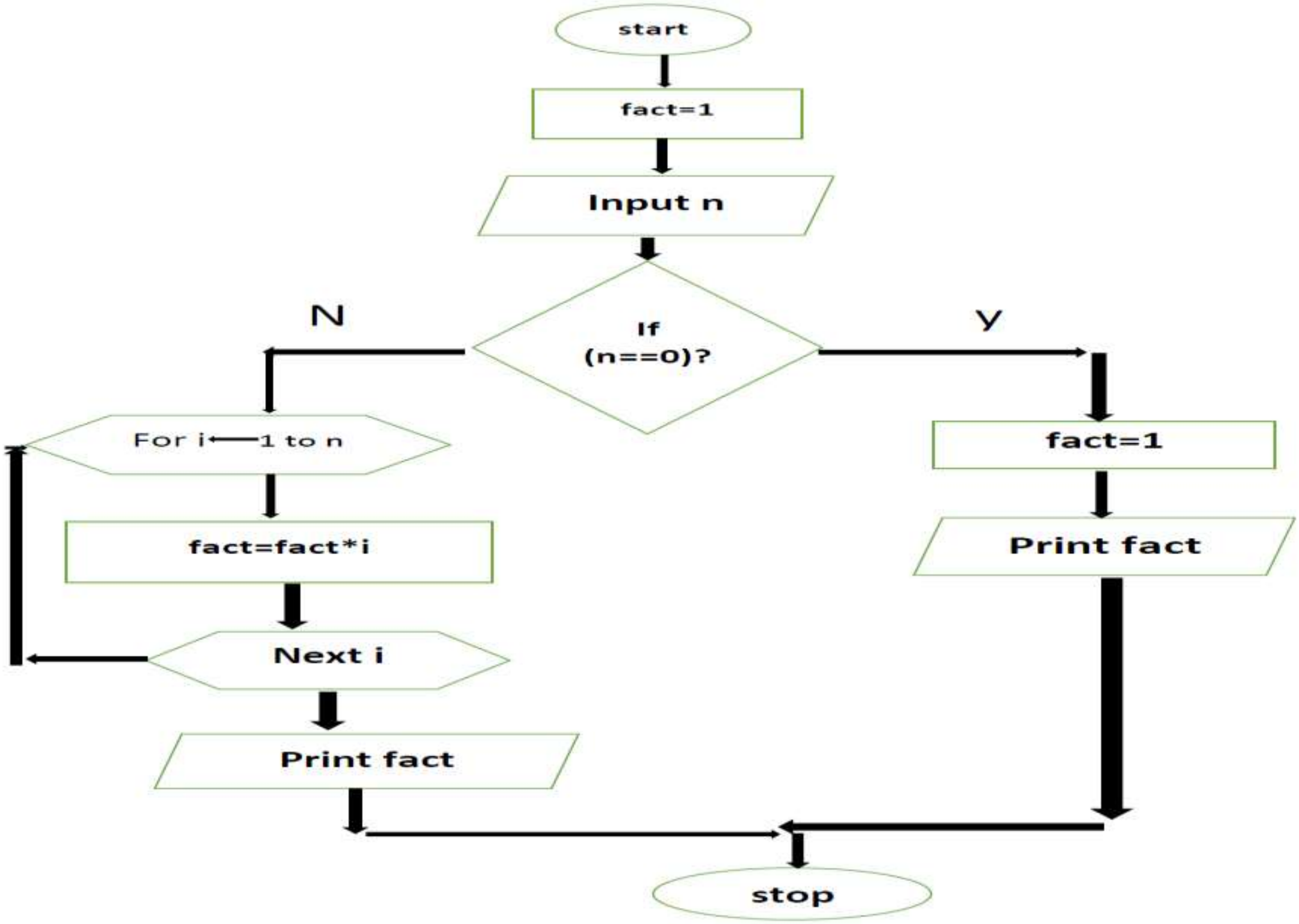

start
input n
sumn=0, sump=0
for i ← 1 to n
input num
Yes
No
if num>0
sump=sump+num
sumn=sumn+num
next i
print sum
stop

FIND THE FACTORIAL OF N

ALGORITHM

```
1. start
2. input n, factorial =1
3. if (n==0)
       print "factorial=1"
   else
         {
         for    i ⟵ 1 to n
        Factorial ⟵ factorial*i
        }
   print "factorial"
4. stop
```

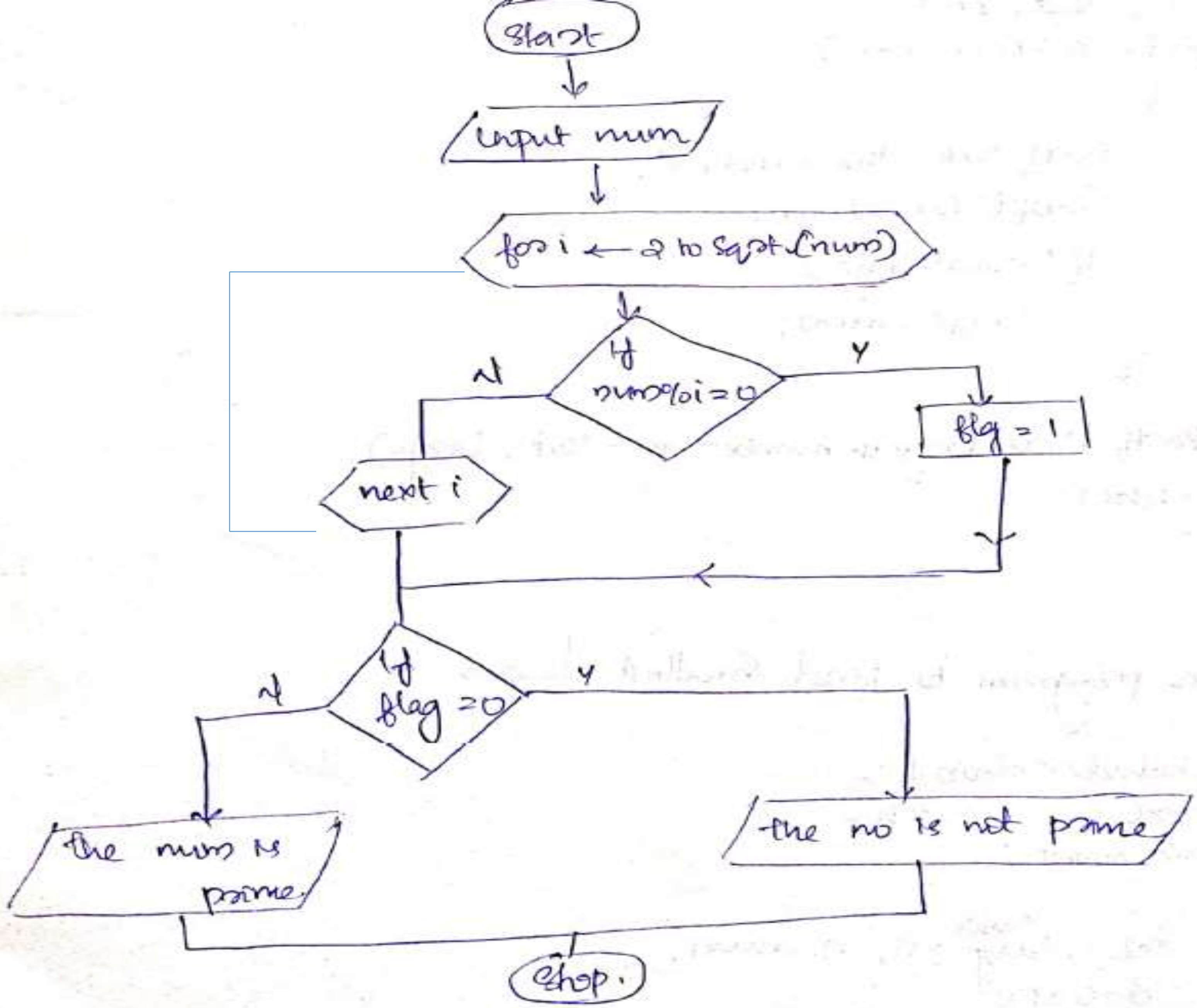
Flow chart


start
fact=1
Input n
If (n==0)?
N
Y
For i ⟵ 1 to n
fact=fact*i
Next i
Print fact
fact=1
Print fact
stop

Check whether a number is prime or not

Algorithm

```
1.  Start
2.  Input num
3.  flag ⟵ 0;
4.  For i ⟵2 to sqrt(num)
        if (num%i==0)
        {
       flag=1;
       goto step 5
        }
5.  if(flag=0)
        print “prime”
    else
       print”Not prime”
6. stop
```


start
input num
for i ← 2 to sqrt(num)
if num%i = 0
Y
N
flg = 1
next i
if flag = 0
Y
N
the num is prime
the no is not prime
stop.

Computer languages

The computer languages can be broadly classified into three categories.

1 . Machine language

2 . Assembly language

3 . High level language

Machine language

- All computers can understand only one language called machine language of the computer.
- Machine language in a string of binary 1‘s and 0‘s. The circuitry of the computer can recognize the machine language instructions immediately. It converts them into electrical signals needed to execute them.
- A machine language instruction has a two part format. The first part is **operation code** that tells the computer what function is to perform and the second part is **operand** that tells where to find or store the data , on which the compiler has to perform the function.

Machine language

- .Every computer has a set of operation codes called its instruction set
- .Each operation code(opcode) is meant to perform a specific basic operation . Typical operations include
- a. Arithmetic Operations
- b . Logical operations
- c . Branch operation
- d . Data movement operation

Machine Languages

- **Advantages of machine language**
  - A computer can execute programs written in machine language at a great speed because the computer can understand machine instructions and hence there is no need for any translation .

- **Disadvantages**
  - Machine dependent
  - Difficult to write a program
  - Error prone
  - Difficulty to modify and correct

2 . Assembly language

- Assembly language programming was introduced in 1952.
- It helped in overcoming the limitations of machine language programmimg to some extent in the following manner.
- By using alphanumeric mnemonic codes for the instructions in the instruction set . Eg : 1110 is used for addition in machine language. Instead of 1110, ADD is used.

Assembly language

- By using alphanumeric names for representing addresses of storage location instead of numeric addresses . Eg: memory address 1000 may be named as first .
- By providing pseudo instructions for instructing the assembler how we want to assemble the program . START PROGRAM at 0000 (It tells the assembler that the instruction of the main program should start at 0000)

Assemblers

- A computer can directly execute only machine language programs
- Hence we have to convert an assembly language program to equivalent machine language program.
- Assemblers are translator programs which does this translation .It is called assembler because in addition to translation it also assembles the corresponding machine language program into the main memory making it ready for execution.
- There is a one to one correspondence between the assembly language and machine language as the assembler translates each assembly language instruction into an equivalent machine language instruction.

Eg : Instead of 1110 for addition assembly level language makes use of ADD

Advantages of assembly language

- Easier to understand
- Easier to locate and correct errors
- Easier to modify
- No worry about addresses
- Easily relocatable
- Efficiency of machine language

Limitations of Assembly language

- Still Machine language
- Knowledge of hardware is required
- Machine level coding

High level language

- They are written in languages which are easy to understand and write because it resembles the English language.
- Both machine and assembly level languages are known as Low level programming languages as they are both machine dependent

Features of High level languages

- They are machine independent
- They do not require programmers to know anything about the internal structure of the computer.
- They do not deal with machine level coding and allow the programmers to write instructions using English words and familiar mathematical models.

Advantages and Limitations of High level language

- **Advantages**
    - Machine independence
    - Easier to learn and use
    - Fewer errors
    - Lower program preparation cost ; ie , less time and effort
    - Better documentation ⍰ Easier to maintain
- **Limitations**
    - Lower efficiency
    - Less flexibility

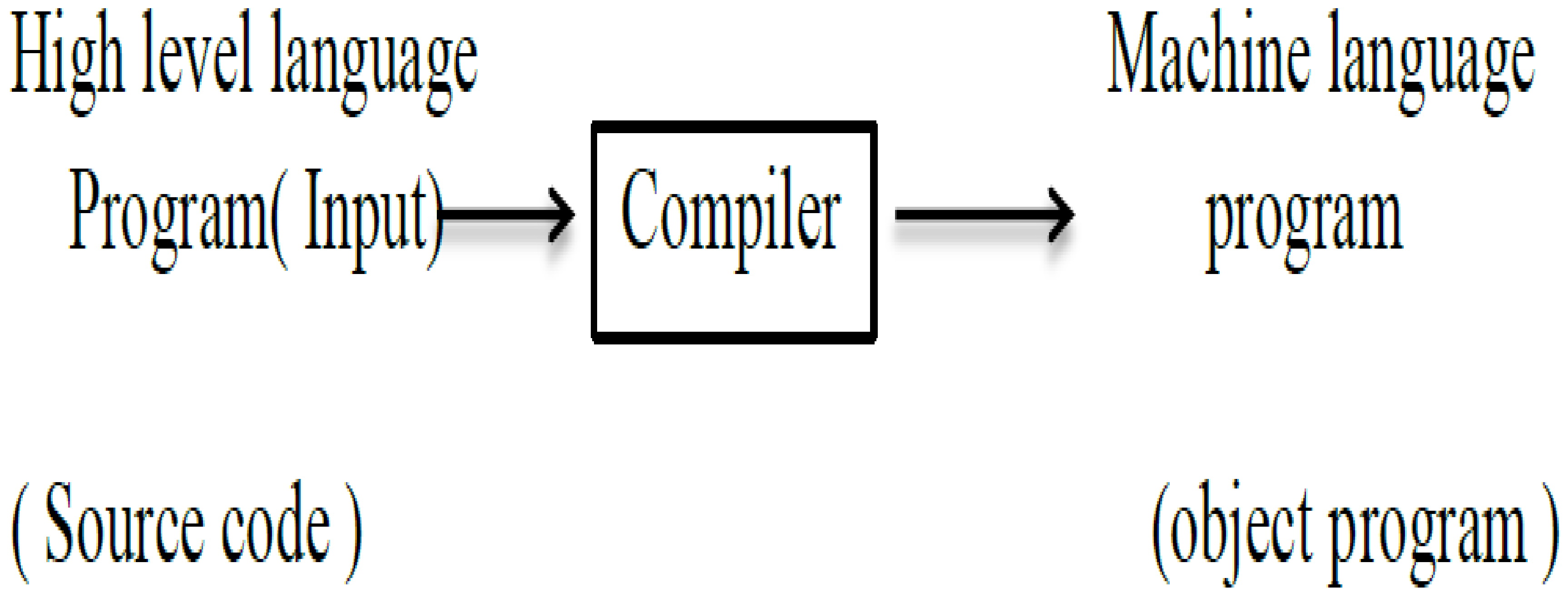
Compiler

- A computer can execute only machine language programs. Thus , high level language program must be converted into its equivalent machine language program .
- The translator program that convert a high level language program into the machine language is called compiler.
- A compiler can only translate only a source program written in a particular language to its corresponding machine level language.
- Eg : A C compiler can only translate C programs into its corresponding machine level language. A COBOL compiler can only translate source programs written in COBOL

Compilers

.

- Compilers are large programs residing permanently on secondary storage
- To translate a source program , a computer first loads the compiler and the source program from the secondary storage into the main memory.
- The computer then executes the compiler with the source programs in its input data.
- It generates the object program which the compiler saves in a file on secondary storage .
- There is no need to repeat compilation process every time we wish to execute .This is because the object program is already in machine language .Compilation is only necessary when we modify the program.


High level language
Program(Input)
Compiler
Machine language
program
(Source code)
(object program)

LINKER

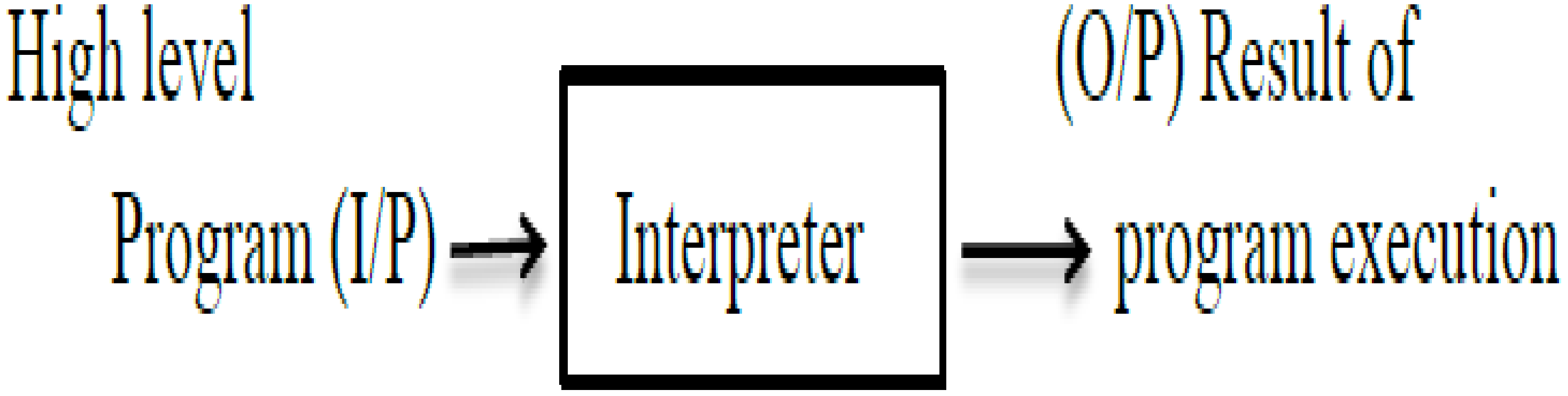
- Large size software consists of several thousands , even millions of lines of program code . For such software , it is almost impractical to store its entire source code in a single file .
- This is because
  - Loading a large size file for compilation on a computer with limiting memory capacity is not possible
  - We need to recompile the entire program whenever we make a small change to the source program .
- To overcome these problems , software developer adapt a modular approach , they divide the software into modules and write separate source program for each module. They convert each of the source program to its corresponding object code.

LINKERS

- We can modify and compile each source program independent of other source program files
- A program called Linker is used to combine all object program files of the software and convert this into final executable program which is sometimes called load module .

INTERPRETERS

- Interpreter is another type of translator used to translate high level language into its equivalent machine language program.
- It takes one statement of the high level language program , translates it into machine language and executes it the machine language instructions immediately.
- It differs from a compiler where the entire source program is translated into object program and is not involved in execution .
- An input to an interpreter is a source program but unlike a computer, the output is the result of program execution instead of object program .


High level
Program (I/P)
Interpreter
(O/P) Result of
program execution

Difference between compiler and interpreter

Compilers	**Interpreters**
• After compilation a user saves the resulting object program for future use and uses it every time he has to execute the program. Hence, repeated compilation is not necessary for repeated execution of the program	• As the user does not save the object program repeated interpretation of a program is necessary for repeated execution
• Compilers translates the entire source program into the corresponding object program	• Interpreter translate each statement and executes it immediately
• Compilers are difficult to work with and have complex programs	• Interpreter are easier to write and have less complex programs
• They require more memory space.	• They require less memory space

Difference between compiler and interpreter

Compilers

- Compiler shows the syntax error only after compiling the source program
- Compilers are faster . Computers can execute saved object file every time it executes it

Interpreters

- Interpreter shows a syntax syntax error only after compiling the source program as soon as it interprets a program statement . This allows the programmer to make corrections during the program development.
- Interpreters are slower than compilers when running a finished program. This is because an interpreter translates each statement from the source program every time it executes it

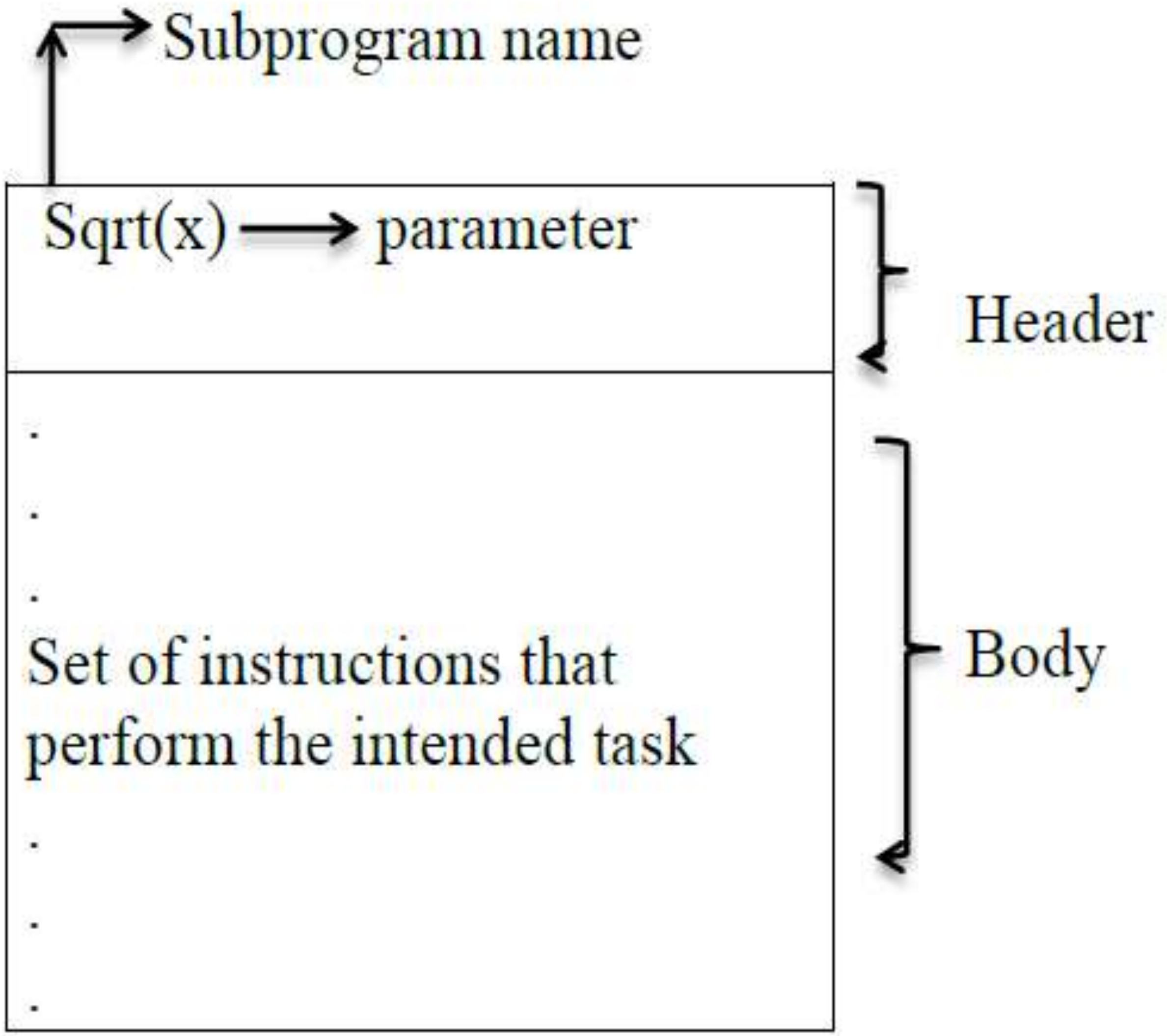
Factors for selecting a language

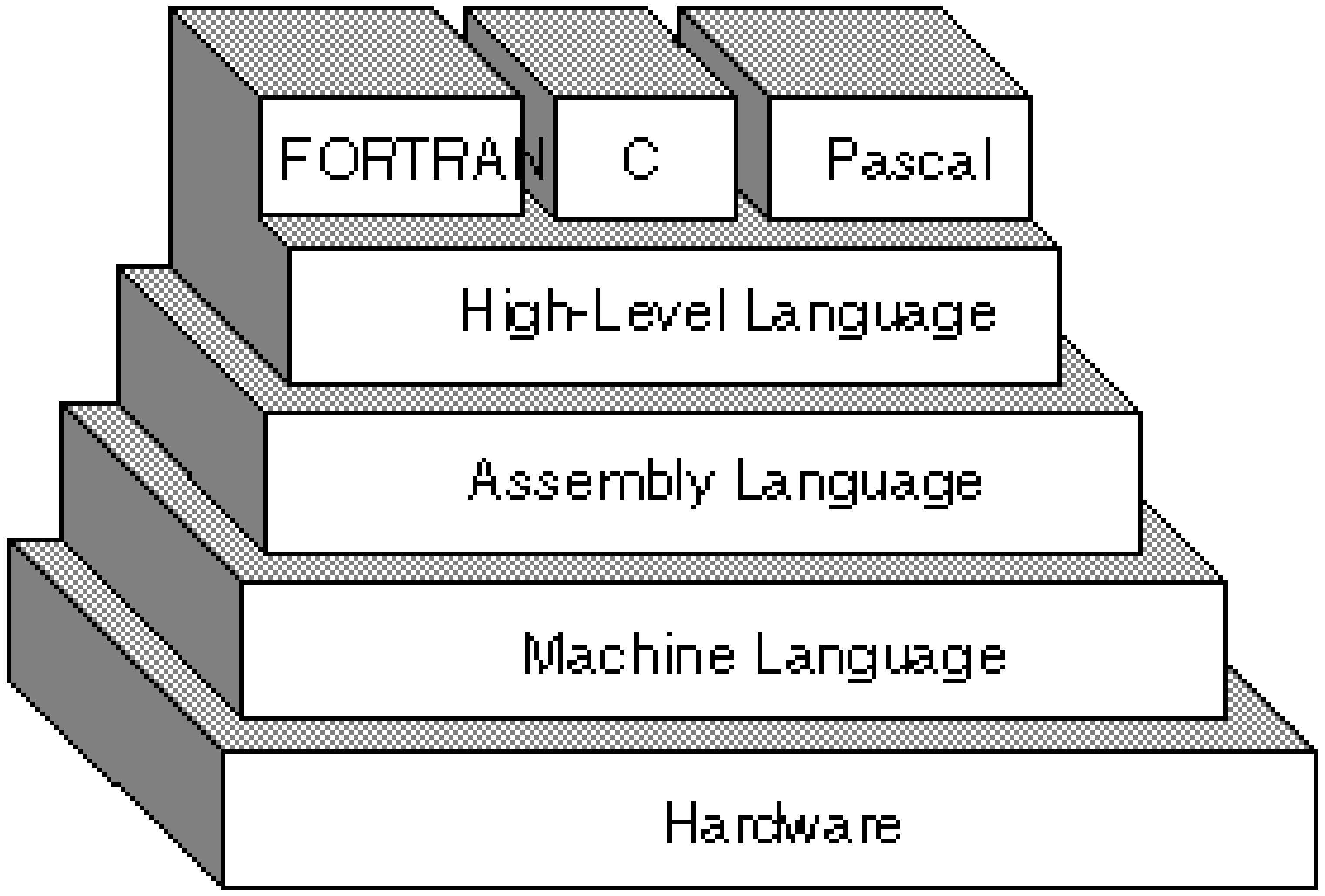
- Nature of the application
- Familiarity
- Ease of learning the language
- Availability of program development tools
- Execution efficiency
- Features of a programming language

Subprogram

- A Subprogram is a program written in a manner that a programmer can use it in other programs whenever needed without a rewriting .
- Subprograms are also called subroutines , sub- procedures and functions.
- There are two types of subprograms- Built in functions and user defined subprograms.

Structure of a subprogram




FORTRAN
C
Pascal
High-Level Language
Assembly Language
Machine Language
Hardware