Module 1.1 to 1.3 **Introduction to Structure Programming Methodology**

INTRODUCTION

- Today, computers are all around us.
- We use them for doing various tasks in a faster and more accurate manner.
- For example, using a computer or smartphone, we can book train tickets online.

Case Study of computerization : Railway Reservation

- Railway reservation is a complex task.
- Online booking of train tickets has added to our comfort by enabling us to book tickets from anywhere, anytime.
- Making reservation involves information about many aspects, such as:
  - details of trains
    - train type,
    - types of berth and compartments in each train,
    - their schedule,etc.,
  - simultaneous booking of tickets by multiple users etc.
- It is only due to the use of computers that today, the booking of the train tickets has become easy.

Computerization

- Computerization indicates the use of computer to develop software in order to automate any routine human task efficiently.
- Computers are used for solving various day-to-day problems.
- Thus problem solving is an essential skill that a computer science student should know.

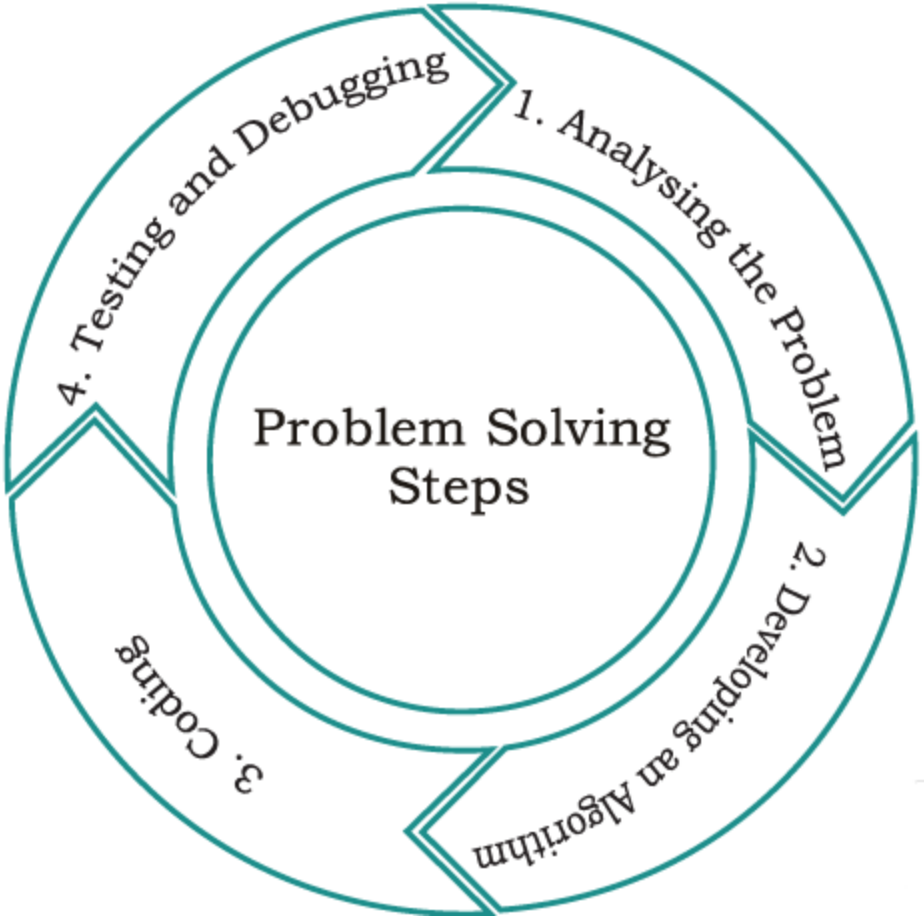
Some other real world problems

- Searching using a search engine,
- sending a message,
- finding a word in a document,
- booking a taxi through an app,
- performing online banking,
- playing computer games

Towards Problem Solving

- Computers themselves cannot solve a problem.
- Thus, the success of a computer in solving a problem depends on how correctly and precisely we, the coders:
  - define the problem,
  - design a solution (algorithm) as a set of step-by-step instructions
  - implement the solution (program) using a programming language and
  - test and improve the program whenever necessary

STEPS FOR PROBLEM SOLVING



Step 1: Analysing the problem

- Clearly understand a problem before designing the solution for it, otherwise we may end up developing a program which may not solve our purpose.
- Read and analyze the problem statement carefully and list the **principal components** of the problem and decide the **core functionalities** that our solution should have.
- Analyze the problem to figure out what are the **inputs** that our program should accept and the **outputs** that it should produce.

Step 2: Developing an Algorithm

- An **algorithm** is a sequence of instructions for computing an answer (**output**) for a problem and data (**input**).
- An algorithm is in a natural language, i.e., a language that humans can easily understand and independent of the programming language.
- Remember that computers are just machines that will execute the instructions as given.
- It is essential to device an algorithm, before you write your program.

Step 2: Developing an Algorithm

- An algorithm is like a very well-written recipe for a dish, with clearly defined steps that, if followed, one will end up preparing the dish as expected.
- We start with a tentative solution plan and keep on refining the algorithm until the algorithm is able to capture all the aspects of the desired solution.
- For a given problem, more than one algorithm may be possible, in such cases we have to select the most suitable solution, based on some criteria like the time constraints, space constraints, accuracy required etc.

Step 3: Coding

- The computer cannot understand the **algorithm** written in natural language that we have finalized in the previous step directly.
- We need to convert the algorithm into a **program**, which is written in a **programming language** that can be understood by the computer.
- The program will take inputs and generate the desired output.

Low Level Prog. Languages

- Low-Level language is the only language which can be understood by the computer.
- Low-level language is also known as **Machine Language**.
- The machine language contains only two symbols **1 & 0**.
- All the instructions of machine language are written in the form of binary numbers 1's & 0's.
- A computer can directly understand the machine language.

High Level Prog. Languages

- High-level language is a computer language which can be understood by the users.
- The high-level language is very similar to human languages and has a set of grammar rules that are used to make instructions more easily.
- Every high-level language has a set of predefined words known as Keywords and a set of rules known as Syntax to create instructions.
- The high-level language is easier to understand for the users but the computer can not understand it.

Step 3: Coding

- Different high level programming languages can be used for writing a program.
  - Examples: Python, C, C++, Java
- It is equally important to record the details of the coding procedures followed and document the solution.
- This is helpful when revisiting the programs at a later stage.

Step 4 : Testing and Debugging

- The program is tested for syntactical errors.
- The program is tested on various parameters such that it meets the requirements of the user.
- It must respond within the expected time.
- It should generate correct output for all possible inputs.
- In case the output generated is incorrect, then the program should be checked for logical errors, if any.

Step 4 : Testing and Debugging

- Software industry follows standardized testing methods while developing complex applications like:
  - unit or component testing,
  - integration testing,
  - system testing, and
  - acceptance testing
- This is to ensure that the software meets all the business and technical requirements and works as expected.

Step 4 : Testing and Debugging

- Once the software application has been developed, tested and delivered to the user, still problems in terms of functioning can come up and need to be resolved from time to time.
- The maintenance of the solution, thus, involves
  - fixing the problems faced by the user,
  - answering the queries of the user and
  - even serving the request for addition or modification of features.

More on Algorithms and Flowcharts

Why do we need an Algorithm?

- The programmer first prepares a roadmap of the program to be written, before actually writing the code.
- Without a roadmap, the programmer may not be able to clearly visualize the instructions to be written and may end up developing a program which may not work as expected.
- Such a roadmap is nothing but the algorithm which is the building block of a computer program.
- If the algorithm is correct, computer will run the program correctly, every time.
- So, the purpose of using an algorithm is to increase the reliability, accuracy and efficiency of obtaining solutions.

Characteristics of a good algorithm

- Precision — the steps are precisely stated or defined.
- Uniqueness — results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- Finiteness — the algorithm always stops after a finite number of steps.
- Input — the algorithm receives some input.
- Output — the algorithm produces some output.

While writing an algorithm, it is required to clearly identify the following:

- The input to be taken from the user
- Processing or computation to be performed to get the desired result
- The output desired by the user

REPRESENTATION OF ALGORITHMS:

- Using their algorithmic thinking skills, the software designers or programmers analyze the problem and identify the logical steps that need to be followed to reach a solution.
- Once the steps are identified, the need is to write down these steps along with the required input and desired output.
- There are two common methods of representing an algorithm —**Flowchart and pseudo code.**
- Either of the methods can be used to represent an algorithm while keeping in mind the following:
  - it showcases the logic of the problem solution, excluding any implementation details
  - it clearly reveals the flow of control during execution of the program

Flowchart

Visual Representation of Algorithms:

- A flowchart is a visual representation of an algorithm.
- A flowchart is a diagram made up of boxes, diamonds and other shapes, connected by arrows.
- Each shape represents a step of the solution process and the arrow represents the order or link among the steps.

The Flowchart

- A graphical representation of the sequence of operations in an information system or program.
  - Information system flowcharts show how data flows from source documents through the computer to final distribution to users.
  - Program flowcharts show the sequence of instructions in a single program or subroutine. Different symbols are used to draw each type of flowchart.

The Flowchart

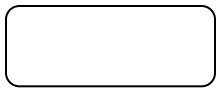
A Flowchart

- shows logic of an algorithm
- emphasizes individual steps and their interconnections
- e.g. control flow from one action to the next

Flowcharts

- Flowcharts is a graph used to depict or show a step by step solution using **symbols** which represent a task.
- The symbols used consist of geometrical shapes that are connected by **flow lines**.
- It is an alternative to pseudocoding; whereas a pseudocode description is verbal, a flowchart is graphical in nature.

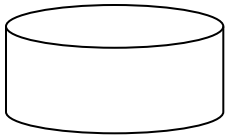
Flowchart Symbols



Terminal symbol - indicates the beginning and end points of an algorithm.

Process symbol - shows an instruction other than input, output or selection.

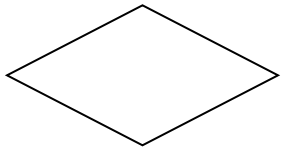
Input-output symbol - shows an input or an output operation.



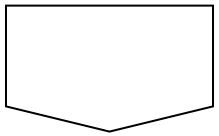
Disk storage I/O symbol - indicates input from or output to disk storage.

Printer output symbol - shows hardcopy printer output.

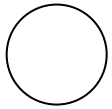
Flowchart Symbols cont…



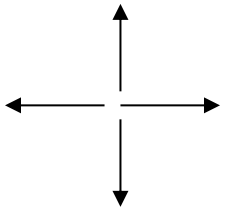
Selection symbol - shows a selection process for two-way selection.



Off-page connector - provides continuation of a logical path on another page.



On-page connector - provides continuation of logical path at another point in the same page.



Flow lines - indicate the logical sequence of execution steps in the algorithm.

Pseudocode

- A pseudocode is another way of representing an algorithm.
- It is considered as a non-formal language that helps programmers to write algorithm.
- It is a detailed description of instructions that a computer must follow in a particular order.
- It is intended for human reading and cannot be executed directly by the computer.
- No specific standard for writing a pseudocode exists.
- The word “pseudo” means “not real,” so “pseudocode” means “not real code”.

Following are some of the frequently used keywords while writing pseudocode:

- INPUT
- COMPUTE
- PRINT
- INCREMENT
- DECREMENT
- IF/ELSE
- WHILE
- TRUE/FALSE

Pseudocode & Algorithm

- **Example 1:** Write an algorithm to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

Pseudocode & Algorithm

Pseudocode:

- *Input a set of 4 marks*
- *Calculate their average by summing and dividing by 4*
- *if average is below 50*

  Print “FAIL”

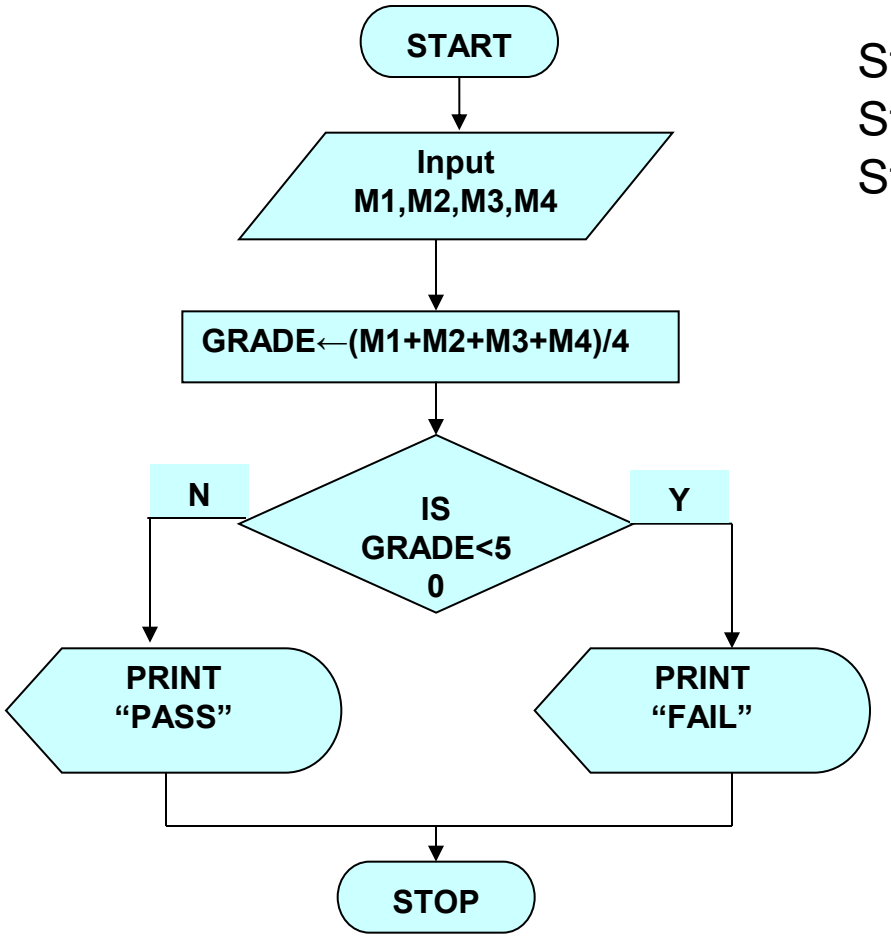
  else

  Print “PASS”

Pseudocode & Algorithm

- Detailed Algorithm
- Step 1: Input M1,M2,M3,M4
  Step 2: GRADE ← (M1+M2+M3+M4)/4
  Step 3: if (GRADE < 50) then
  Print “FAIL”
  else
  Print “PASS”
  endif

Example



Step 1: Input M1,M2,M3,M4
Step 2: GRADE ← (M1+M2+M3+M4)/4
Step 3: if (GRADE <50) then
Print “FAIL”
else
Print “PASS”
endif

Example 2

- Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Pseudocode:

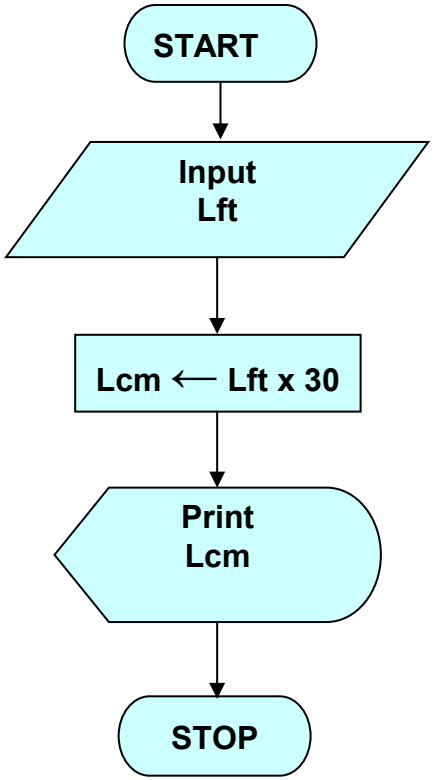
- *Input the length in feet (Lft)*
- *Calculate the length in cm (Lcm) by multiplying LFT with 30*
- *Print length in cm (LCM)*

Example 2

Algorithm

- Step 1: Input Lft
- Step 2: Lcm ← Lft x 30
- Step 3: Print Lcm

Flowchart



Example 3

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

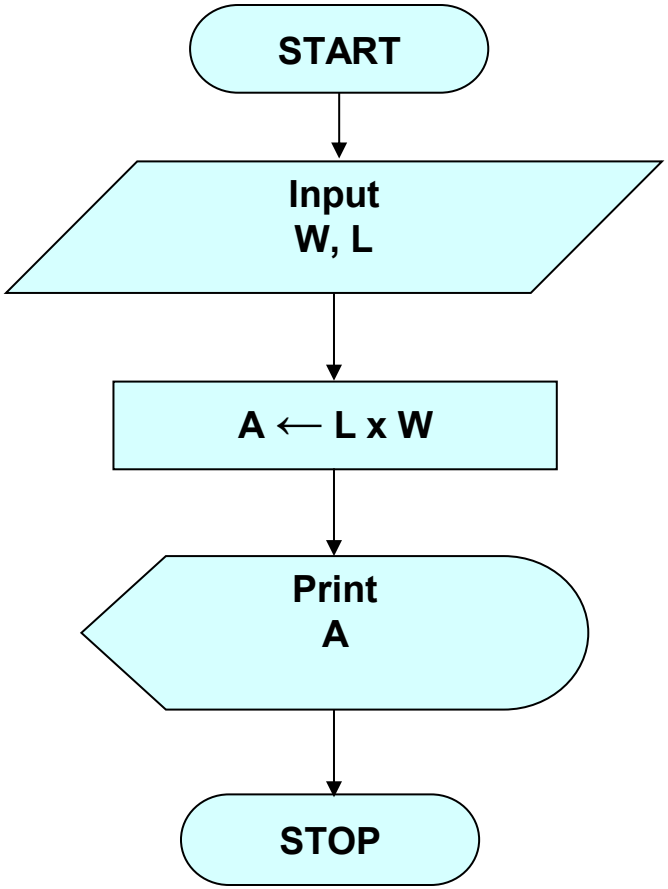
Pseudocode

- *Input the width (W) and Length (L) of a rectangle*
- *Calculate the area (A) by multiplying L with W*
- *Print A*

Example 3

Algorithm

- Step 1: Input W,L
- Step 2: A ← L x W
- Step 3: Print A



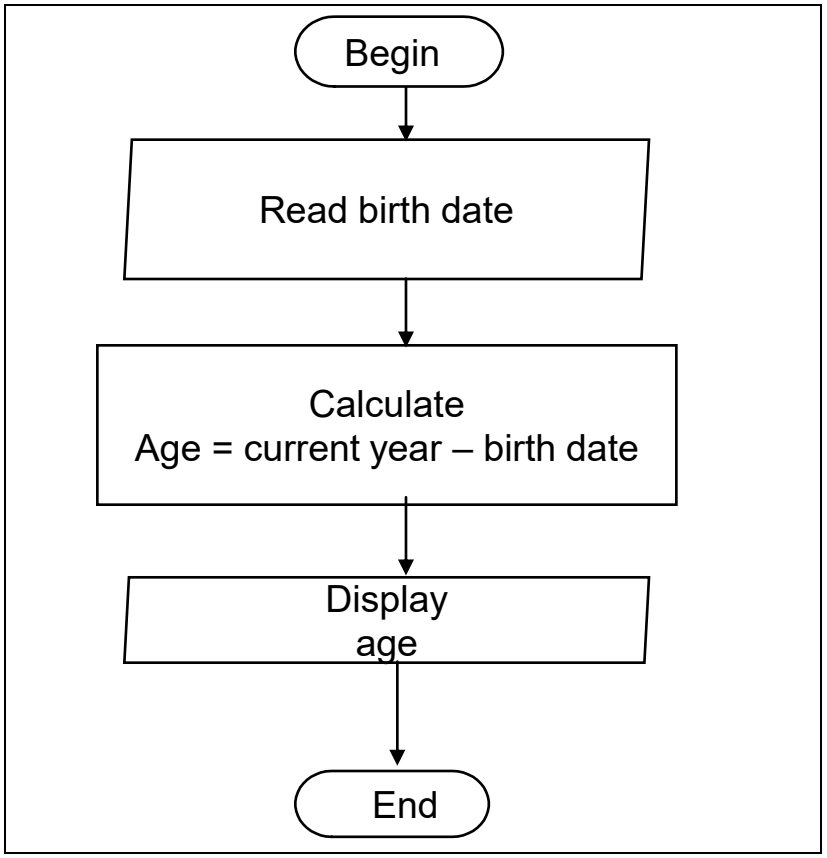
Example 4

- Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation

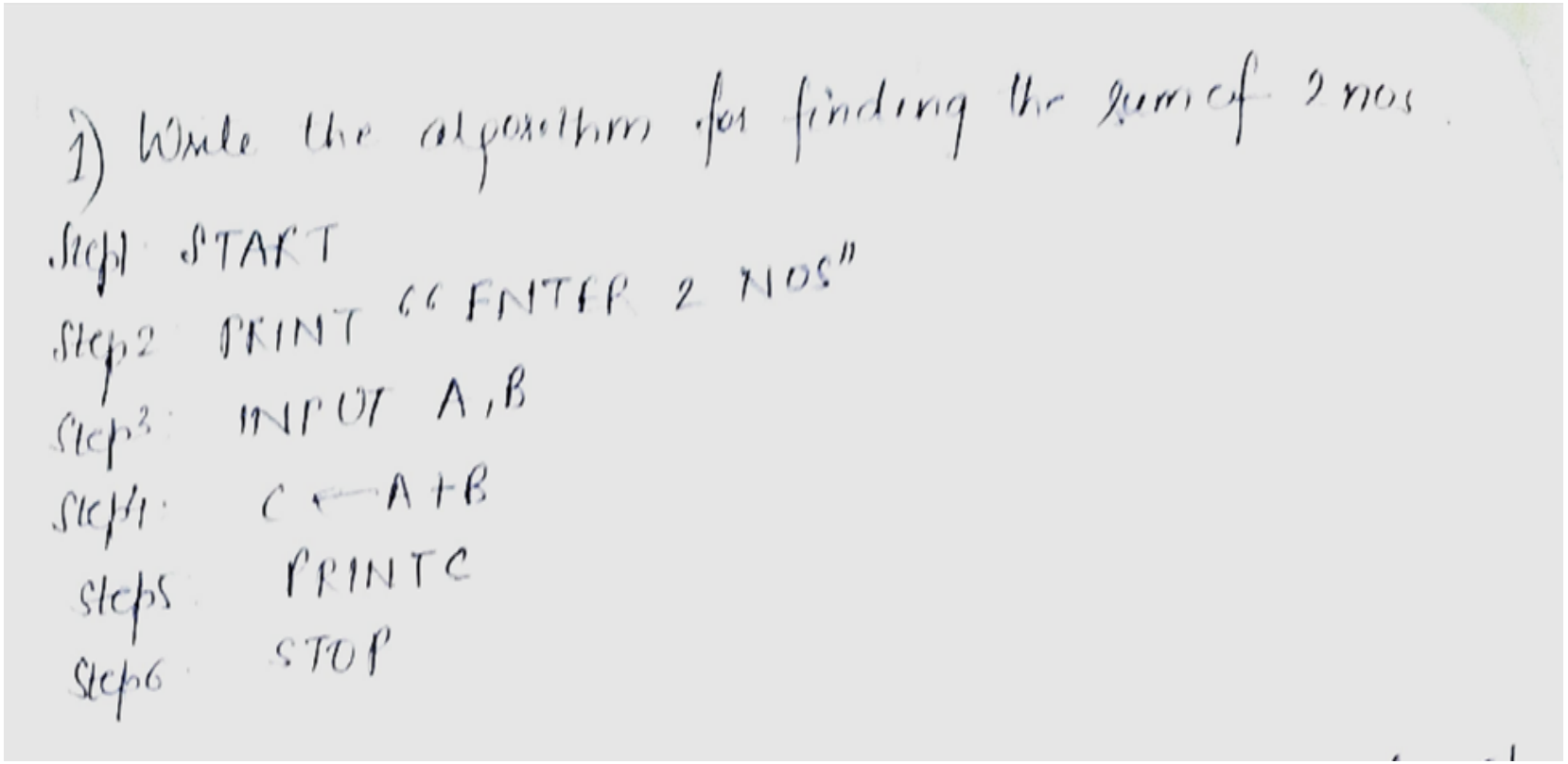
$$ax^2 + bx + c = 0$$

- Hint: **d** = sqrt ($b^2 - 4ac$), and the roots are: ***x*1** = (–*b* + *d*)/2*a* and ***x*2** = (–*b* – *d*)/2*a*

Flowchart – example 1



Algorithm to find sum of two numbers



Algorithm for interchanging the numeric values of two variables

Step1 : START
Step2 : PRINT "ENTER THE VALUE OF A & B"
Step3 : INPUT A, B
Step4 : C ← A
Step5 : A ← B
Step6 : B ← C
Step7 : PRINT A, B
Step8 : STOP

Algorithm for determining remainder

2) Write the algorithm for determining the remainder of a division operation where the dividend & divisor are both integers without using (%) modulus operator.

Step1 : START
Step2 : PRINT "ENTER DIVIDEND"
Step3 : INPUT N
Step4 : PRINT "ENTER DIVISOR"
Step5 : INPUT D
Step6 : Q ← N/D (Integer Division : Only integer value is obtained & remainder ignored)
Step7 : R ← N − Q*D
Step 8 : PRINT R
Step 9 : STOP

the numeric value

Algorithm to check whether a number is odd or even

Write an algorithm to check whether a number given by the user is odd or even.

Step 1. START

Step 2. PRINT "ENTER THE NUMBER"

Step 3: INPUT N

Step 4: R ← N%2

Step 5: IF R=0 THEN
PRINT "N IS EVEN"

ELSE
PRINT "N IS ODD"

Step 6 : STOP

To check whether triangle is equilateral, scalene or isosceles

5)

Step 1 : START

Step 2 : PRINT " ENTER THE LENGTH OF 3 SIDES OF A TRIANGLE"

Step 3 : READ A, B, C

Step 4 : IF A = B AND B = C THEN
PRINT "EQUILATERAL TRIANGLE"

ELSE IF A = B OR B = C OR C = A THEN
PRINT " ISOSCELES TRIANGLE"

ELSE PRINT "SCALENE TRIANGLE"

Step 5 : STOP

To check whether triangle can be drawn or not

Take 3 sides of a Δ and check whether the Δ can be drawn or not.

Step1: START

Step2: PRINT "ENTER THE LENGTH OF THREE SIDES OF A TRIANGLE"

Step3: INPUT A, B, C

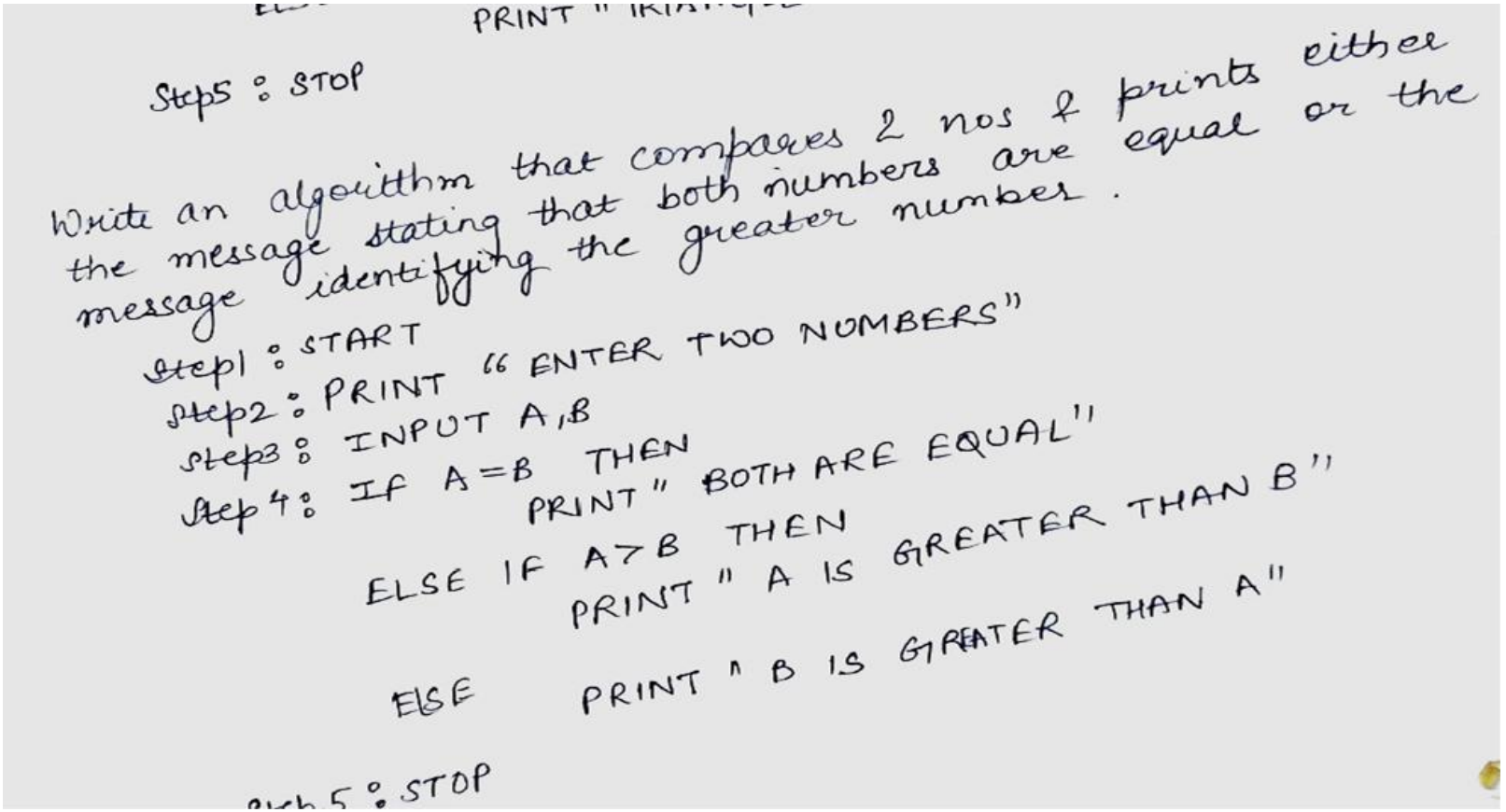
Step4: IF A+B>C AND B+C>A AND A+C>B THEN PRINT "TRIANGLE CAN BE DRAWN"

ELSE PRINT "TRIANGLE CANNOT BE DRAWN"

Step5: STOP

nos & prints either or th

To compare two numbers



PRINT "IRIA...

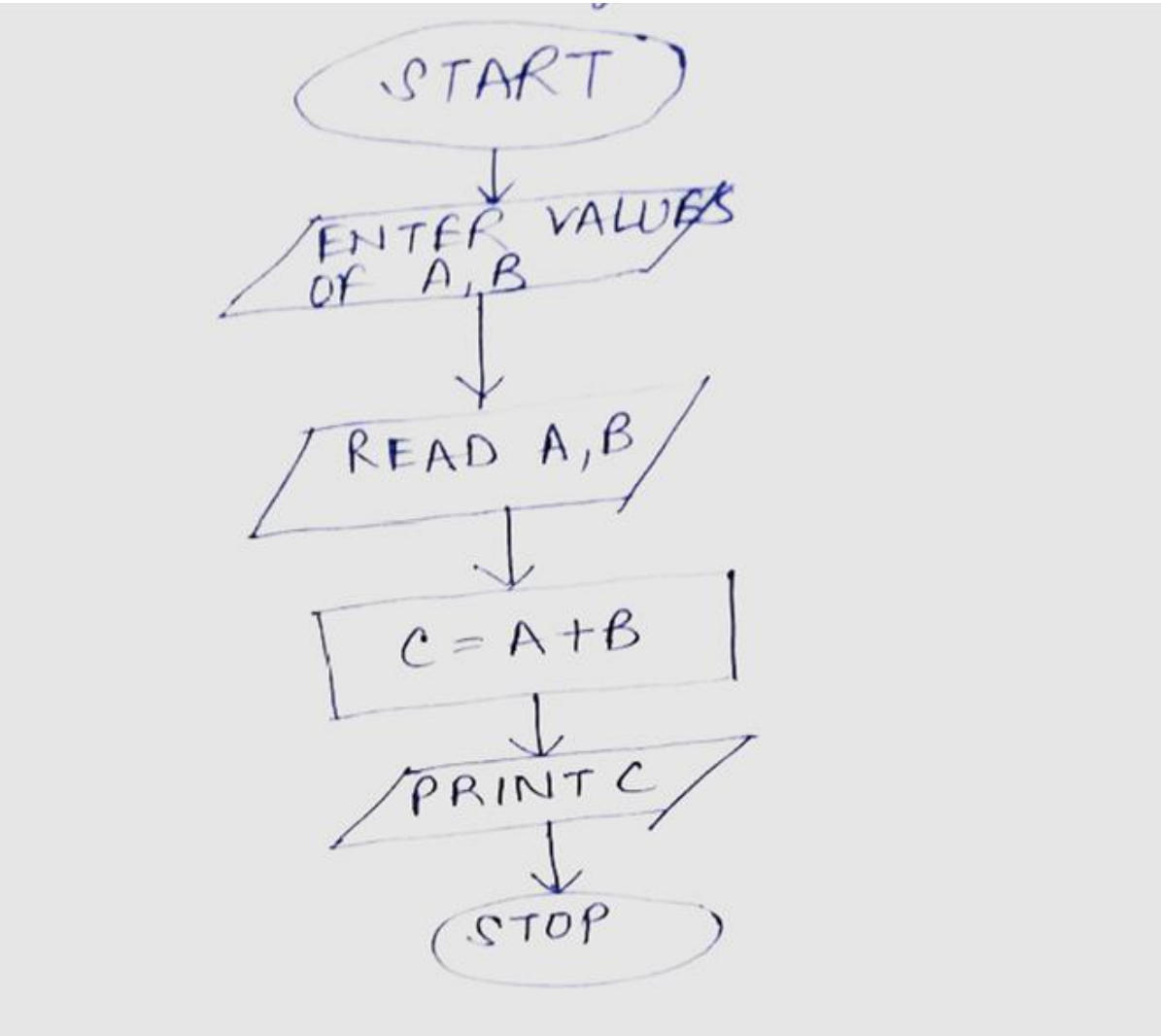
Step5 : STOP

Write an algorithm that compares 2 nos & prints either the message stating that both numbers are equal or the message identifying the greater number.

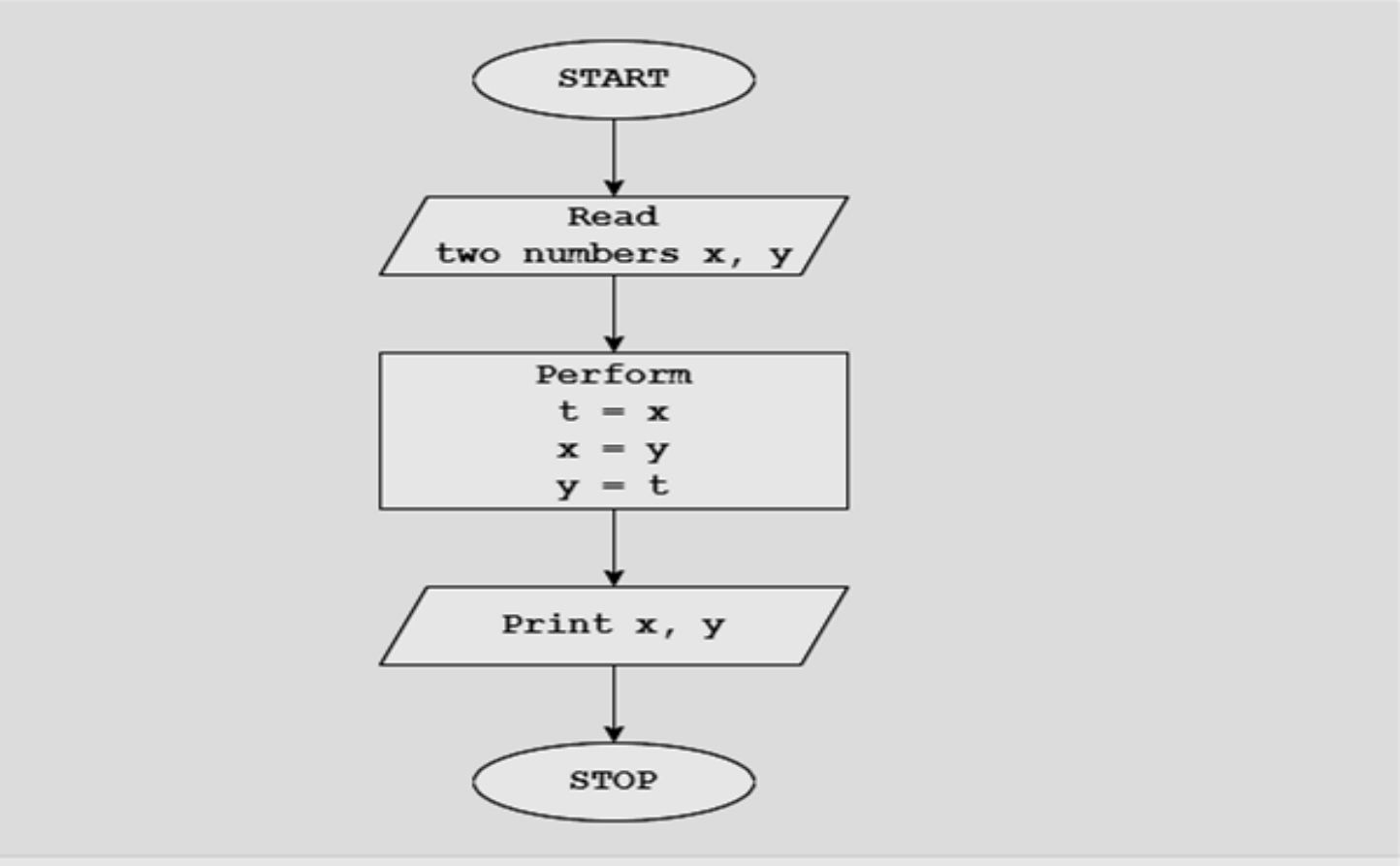
Step1 : START
Step2 : PRINT "ENTER TWO NUMBERS"
Step3 : INPUT A,B
Step4 : IF A=B THEN
PRINT "BOTH ARE EQUAL"
ELSE IF A>B THEN
PRINT "A IS GREATER THAN B"
ELSE PRINT "B IS GREATER THAN A"

Step5 : STOP

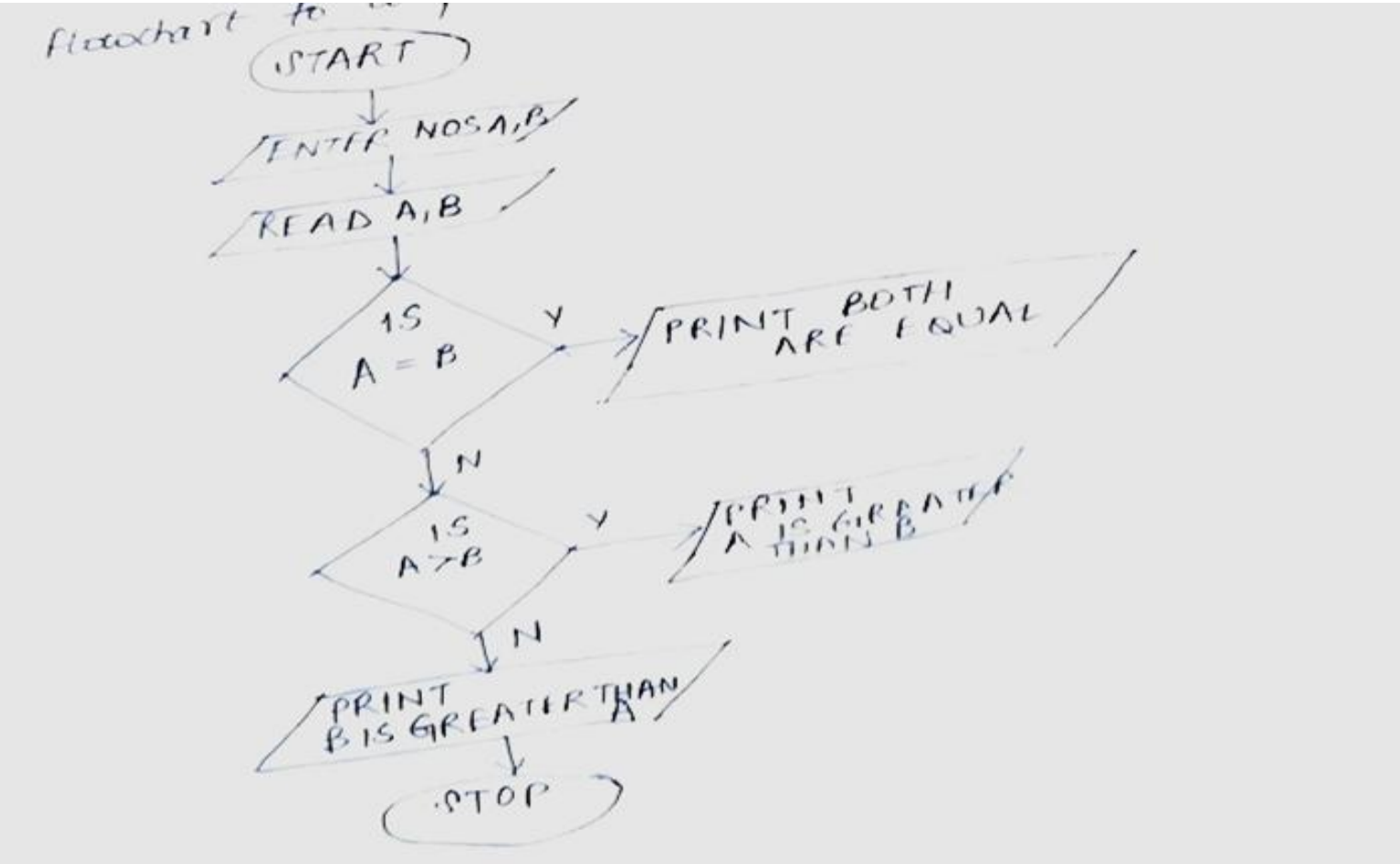
Flowchart to add two numbers entered by the user



Flowchart to swap two numbers using third variable



Flowchart to compare two numbers

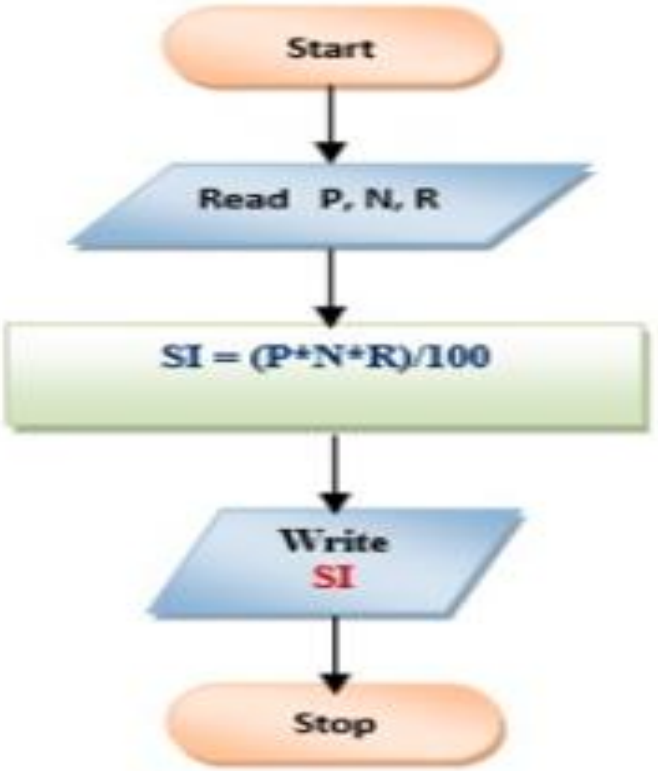


Flowchart to calculate SI

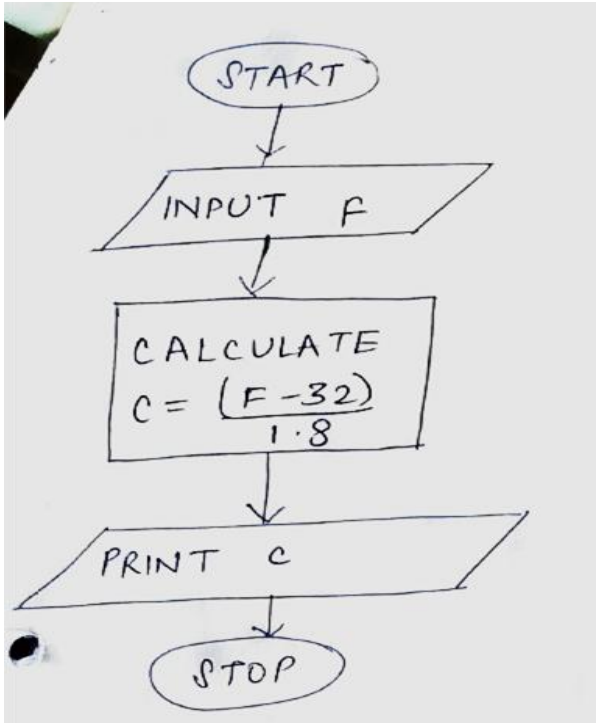
Algorithm

1. Start
2. Read P, N, R
3. SI=(PNR)/100
4. Print SI
5. Stop

Flowchart



Flowchart to convert temperature from Fahrenheit to centigrade



ALGORITHM –

Step 1 : START

Step 2 : PRINT " ENTER VALUE OF TEMP IN FAHRENHIETS"

Step 3 : INPUT F

Step 4 : C = (F-32)/1.8

Step 5 : PRINT C

Step 6 : STOP

Flowchart to print 1 to 20 numbers

Problem: Print 1 to 20 numbers

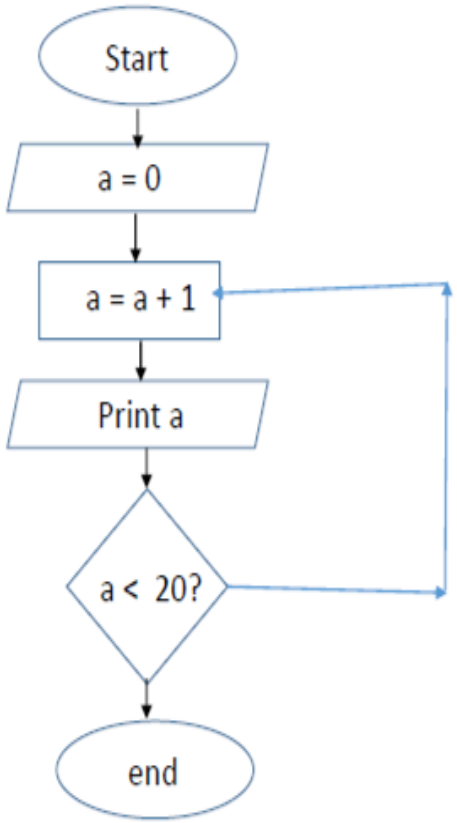
Algorithm:

Step 1: Initialize variable “a” = 0;

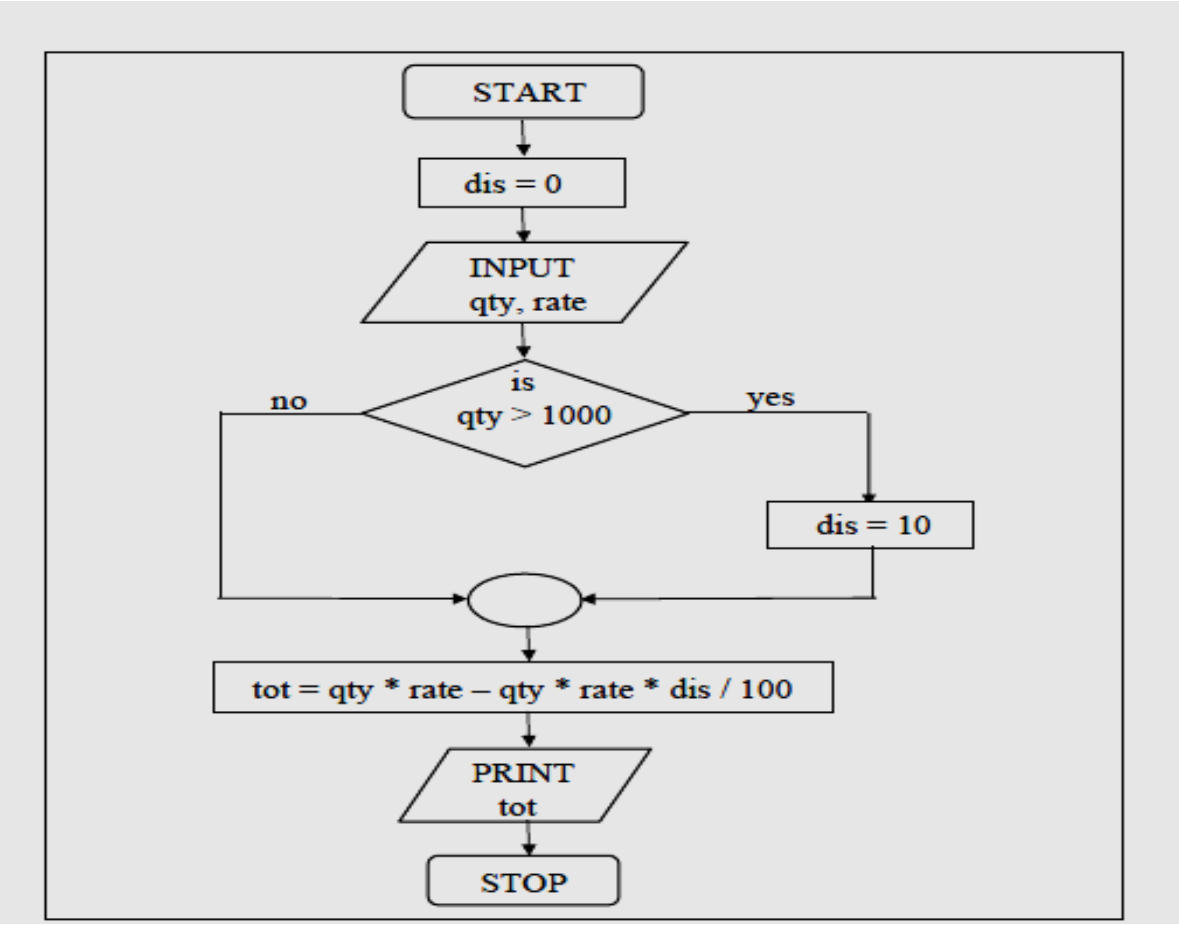
Step 2: Increment “a”

Step 3: Print “a”

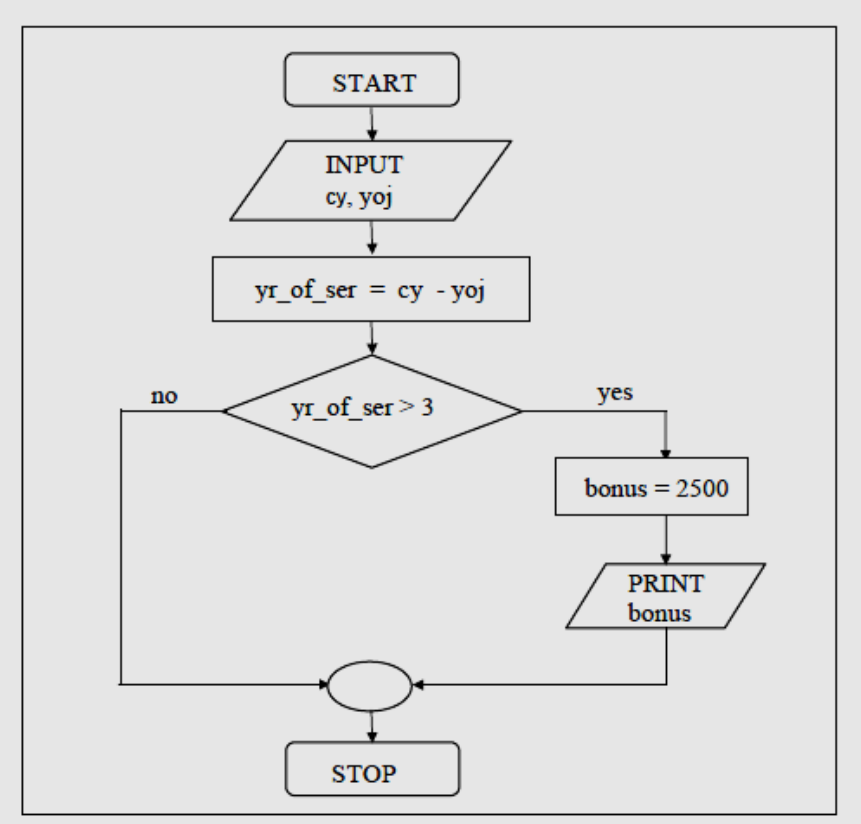
Step 4: Check “a” is less than 20, goto Step 2



While purchasing certain items, a discount of 10% is offered if the quantity purchased is more than 1000. If quantity and price per item are input through the keyboard, Draw the flowchart to calculate the total expenses.

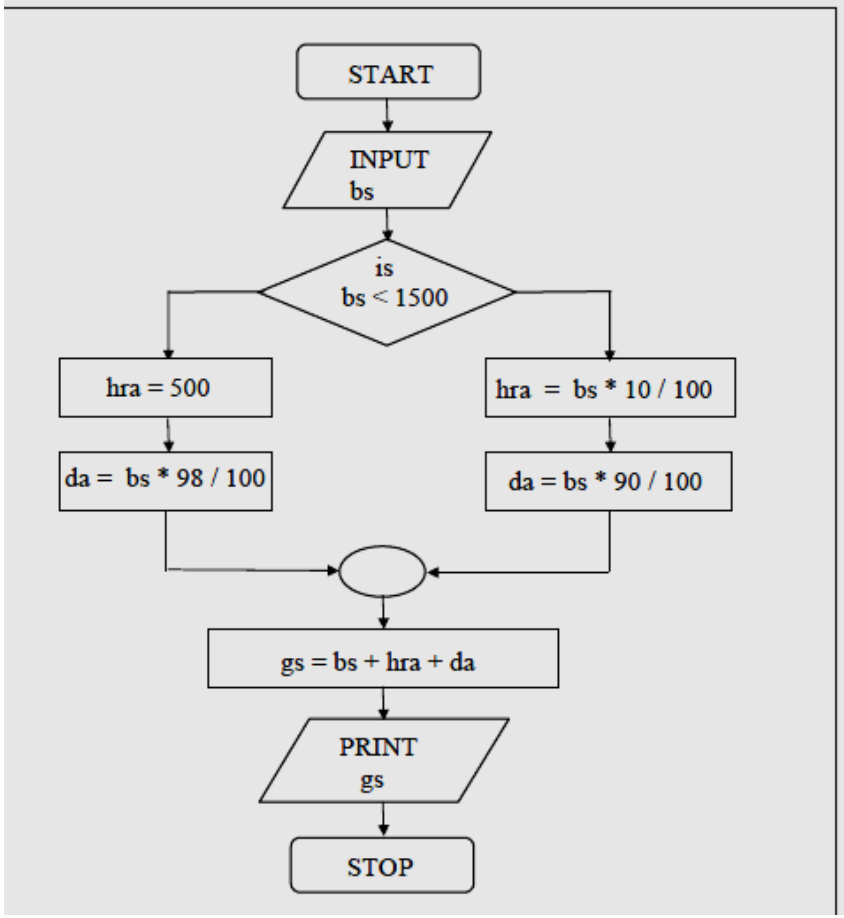


The current year and the year in which the employee joined the organization are entered through the keyboard. If the number of years for which the employee has served the organization is greater than 3 then a bonus of Rs. 2500/- is given to the employee. If the years of service are not greater than 3, then the program should do nothing.



In a company an employee is paid as under: If his basic salary is less than Rs. 1500, then HRA =10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary. If the employee's salary is input through the keyboard write a program to find his gross salary.

In a company an employee is paid as under: If his basic salary is less than Rs. 1500, then HRA =10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary. If the employee's salary is input through the keyboard write a program to find his gross salary.



Exercises: Algorithm & Flowchart

1.) Create an algorithm and a flowchart that will accept/read two numbers and then display the bigger number.

Exercises: Algorithm & Flowchart

2.) Create an algorithm and a flowchart that will compute the area of a circle.

Exercises: Algorithm & Flowchart

3.) Create an algorithm and a flowchart that will compute the sum of two numbers. If the sum is below or equal to twenty, two numbers will be entered again. If the sum is above 20, it will display the sum.

Lab Activity: Algorithm & Flowchart

4) Create an algorithm and a flowchart that will output the largest number among the three numbers.

Structured Programming Methodology

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making specific disciplined use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Following the structured program theorem, all programs are seen as composed of three control structures:

"**Sequence**"; ordered statements or subroutines executed in sequence.

"**Selection**"; one of a number of statements is executed depending on the state of the program. This is usually expressed with keywords such as if..then..else..endif. The conditional statement should have at least one true condition and each condition should have one exit point at max.

"**Iteration**"; a statement or block is executed until the program reaches a certain state, or operations have been applied to every element of a collection. This is usually expressed with keywords such as while, repeat, for or do..until. Often it is recommended that each loop should only have one entry point (and in the original structural programming, also only one exit point, and a few languages enforce this).

Program Execution Process