KSU CET

S1 & S2 Notes

2019 Scheme

Manifest the leadership in you

30/6/2020

MODULE-1

STRUCTURED PROGRAMMING

⇒ * is a method for designing and coding programs in a systematic, organized manner.

* best known programming method.

3 types or forms

* Simplest form of a program is one after the other arrangement or a sequence — Program as a list of instructions that are obeyed one after the other.

* Selection! - Instructions are selected depending on some condition.

* Looping: Instructions are repeated.

⇒ Sequencing, selection and looping are the basic building blocks that makes up a structure program

* S.P controls the complexity of the flow by using only a small number of simple and standard control structures.

— Eg if, if else and switch case for selection

while, do-while, and for for looping.

* A complex prblm is divided into logically independent units and these units are then joined using basic structures -

* Advantages:-
- easier to understand because modules can be considered independently.
- easier to identify the structure of a particular program component.
- easier to maintain or modify (as individual modules can be corrected without changing the entire program).
- easier to design because the modules can be designed independently.

PROBLEM SOLVING

steps involved are:-

1] Problem Analysis
* studying the problem in detail.
* Reducing the problem (restarting)
* Identifying input available, output requirements and constraints in the problem.

2] Algorithm Development & Program Design
* Generating alternate ways of solution
* selecting the best method among the alternatives.
* Preparing a list of procedures or steps necessary for determining the solutions.

3] Computing the results.

* Program coding
* Program compilation and execution.
* Program testing (test cases) and debugging (finding problems).
* Program documentation.

ALGORITHM

(It must be as detailed as possible)

For designing a code
Algorithm
flowchart
pseudocode

* The design specifies how the problem is solved and this description is called algorithm.
* consists of a number of elementary operations and instructions about how the operations are carried out.
* These are usually several alternative algorithm for solving a problem and we select the best one.
* Algorithm can be written in different ways
  * using natural language - common method.
  * Flowchart - pictorial representation of algorithm

⇒ Qualities of a good algorithm

* Input and output should be defined precisely.
* Each step in the algorithm should be clear and unambiguous. (without confusions)
* Algorithms should be most effective among many different ways to solve a problem.
* An algorithm shouldn't have a computer code. Instead, it should be written in such a way that it can be used in different programming languages.

3/07/2020

ALGORITHM

Example 1

Sequence

1] Algorithm to find the sum of two numbers.

Step 1 : start.

Step 2 : Read two numbers x and y

Step 3 : Let $z = x+y$

Step 4 : Display the sum z.

Step 5 : stop.

Example 2

2] Algorithm to evaluate the equation ax^2+bx+c, where x is a variable and a, b, c are coefficients.

Step 1 : start

Step 2 : Read the values of the coefficients a, b, and c

Step 3 : Read the value of x.

Step 4 : Calculate $s = (a*x*x)+(b*x)+c$

Step 5 : Display the results

Step 6 : stop

SELECTION

Example 3

3] Algorithm to find the largest of two numbers.

step1 : start.
step2 : Read two numbers x and y.
step3 : if x > y, then go to step 4, otherwise go to step 6
step4 : Display 'x' as the larget, go to step6.
step5 : Display y as the largest.
step6 : stop.

Example 4

4] Algorithm to find solution of a quadratic equation $ax^2+bx+c=0$.

step1 : start
step2 : Read the coefficients a, b and c
step3 : Calculate d = sqrt((b*b) - (4*a*c))
step4 : If (d=0), then go to step 5, otherwise go to step <6>.
step5 : display that both roots are equal and = -b/(2*a), go to step 10.
step6 : If (d>0), then go to step 7, otherwise go to step 9.

Step7 : Calculate $r_1 = (-b+d)/(2*a)$ and
$r_2 = (-b-d)/(2*a)$.

Step8 : Display roots r_1 and r_2, go to step #10.

Step9 : Display that the equation has imaginary roots

Step10 : Stop.

LOOPING

Examples

5] Algorithm to display numbers from 1 to n.

Step1 : Start

Step2 : Read the value of n

Step3 : Let i=1.

Step4 : Repeat steps 4 to 5 if 'i' is less than n.

Step5 : Display the value of i.

Step6 : Increment i.

Step7 : Stop.

Example 6

6] Algorithm to find the factorial of a number

step1 : start.
step2 : Read a number, num.
step3 : Let i=1 and f=1.
step4 : Repeat steps 5 to 6 if i is less than or equal to num.
step5 : Calculate f=f*i.
step6 : increment i.
step7 : Display the factorial, f.
step8 : stop.

Example 7

7] Write an algorithm to find ~~and~~ display fibonnacci series.

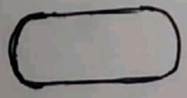
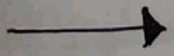
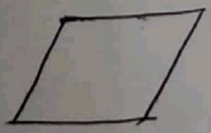
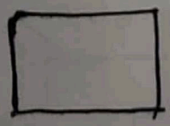
step1 : start.
Step2 : Read the limit a number, num.
step3 : ~~Initialize sum~~, let s1=0 and s2=1.
Step4 : Let i=1
step5 : Repeat steps 6 to 9 if i less than or equal to num.
step6 : Display the value of s1.
step7 : Let sum = s1+s2;
step8 : then s1=s2 and s2=sum.
step9 : Increment i.

Step 10: Stop.

7/07/2020

FLOWCHART

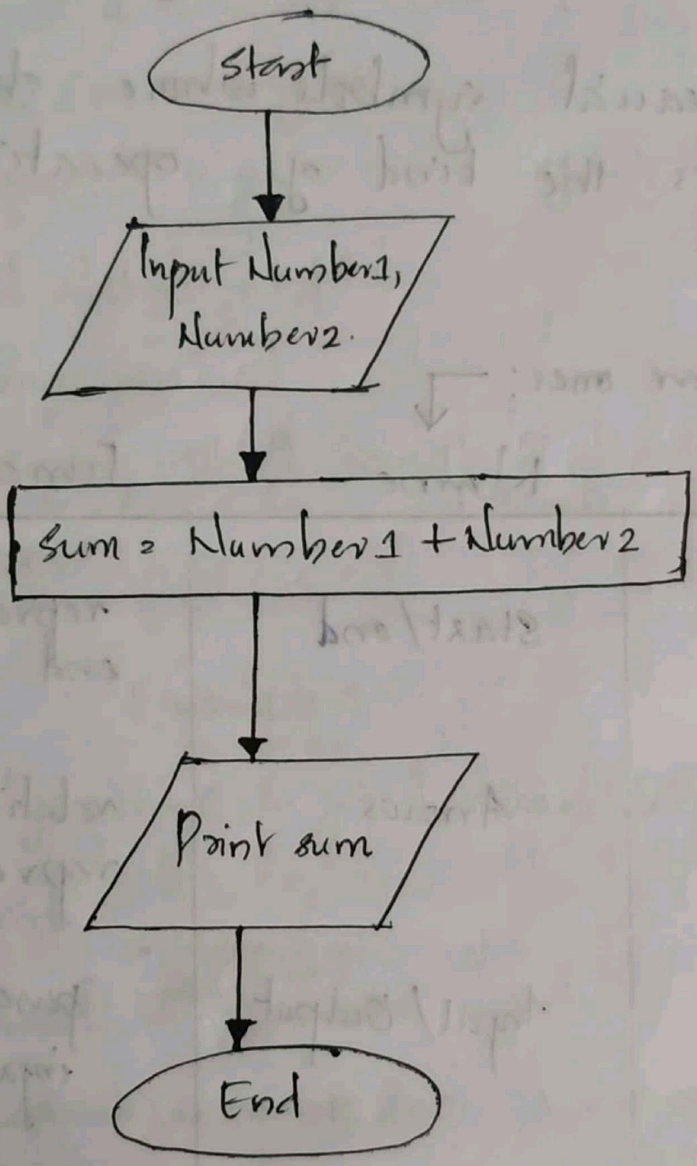
* Graphical representation of algorithm.
* Uses speacial symbols whose shape determines the kind of operation being performed.
* Most popular ones:

Symbol	Name	Function.
	Start/end	represents start and end
	Arrows	relationship btw the representative shapes.
	Input/Output	parallgm represents input or output
	Process	represents a process
	Decision.	A diamond indicates a decision (or condition)

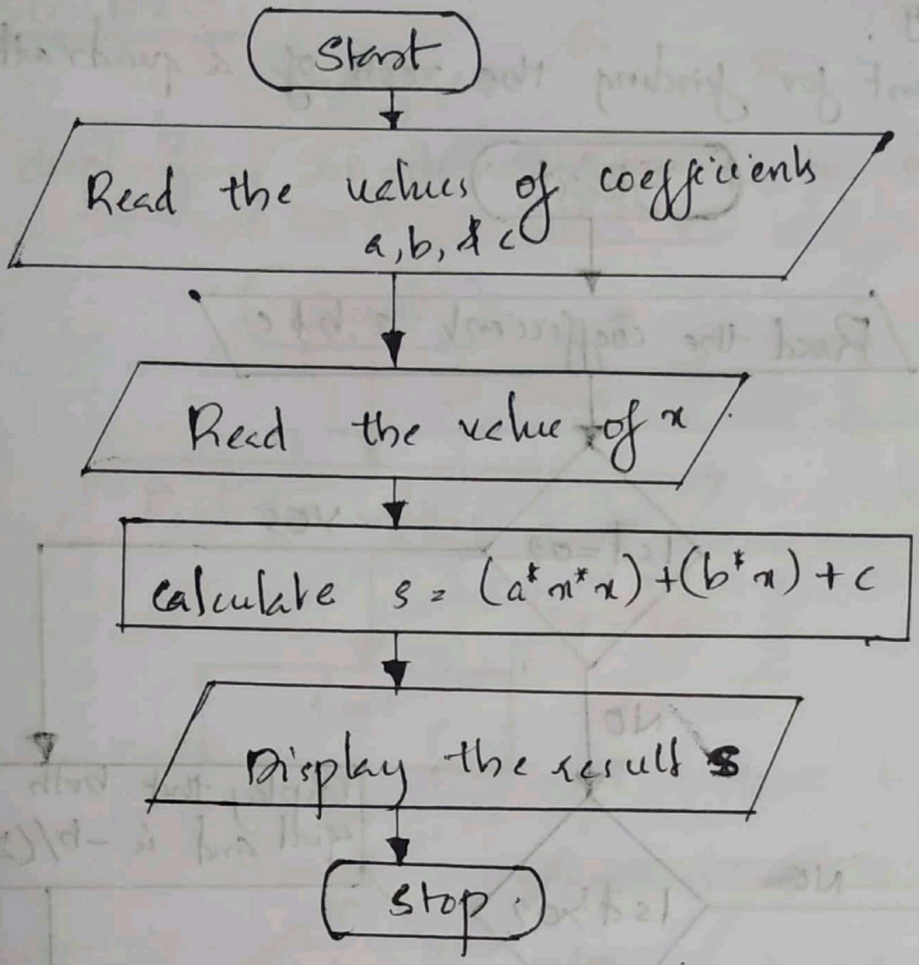
→ SEQUENCE

Example - 1

1] Flow chart for finding sum of two numbers.



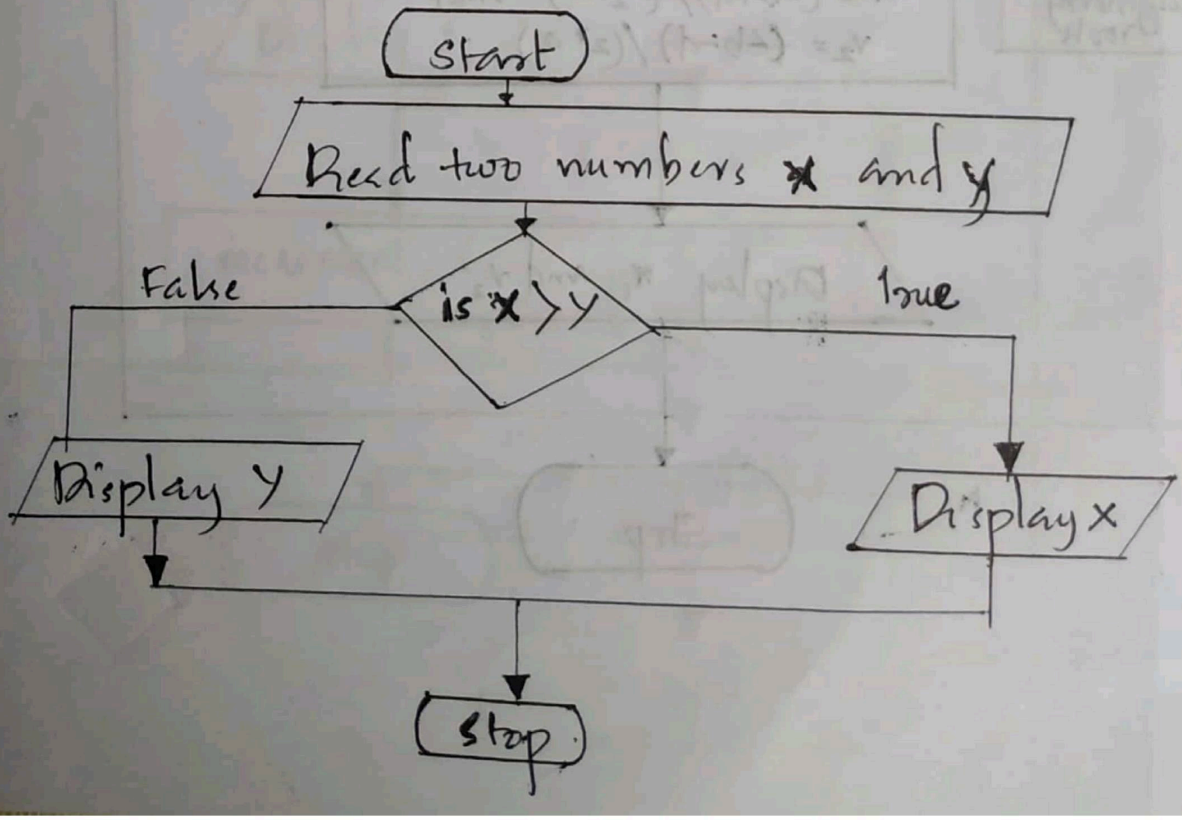
Example - 2



→ SELECTION

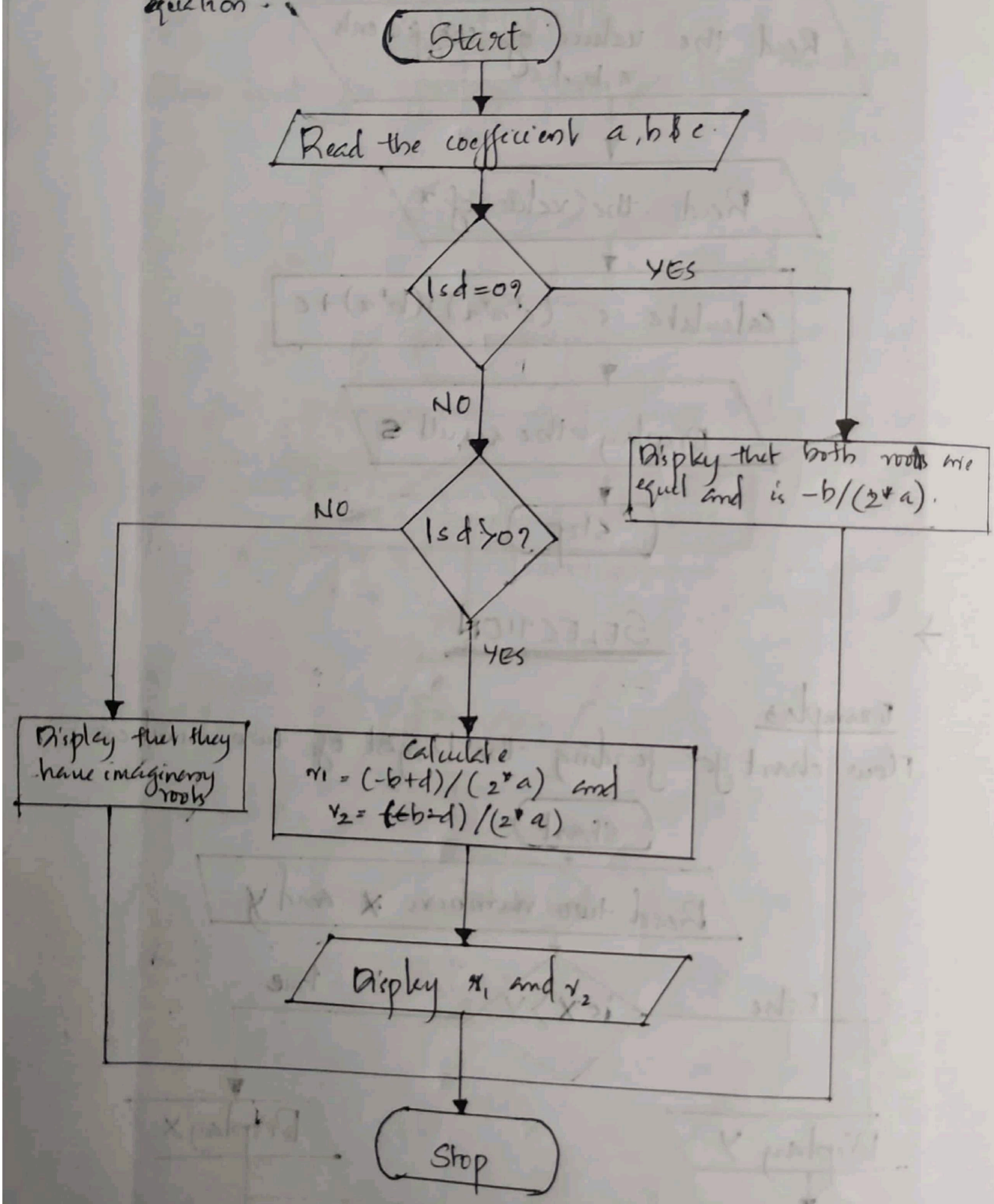
Example 3

- Flow chart for finding the largest of two numbers.



Example:

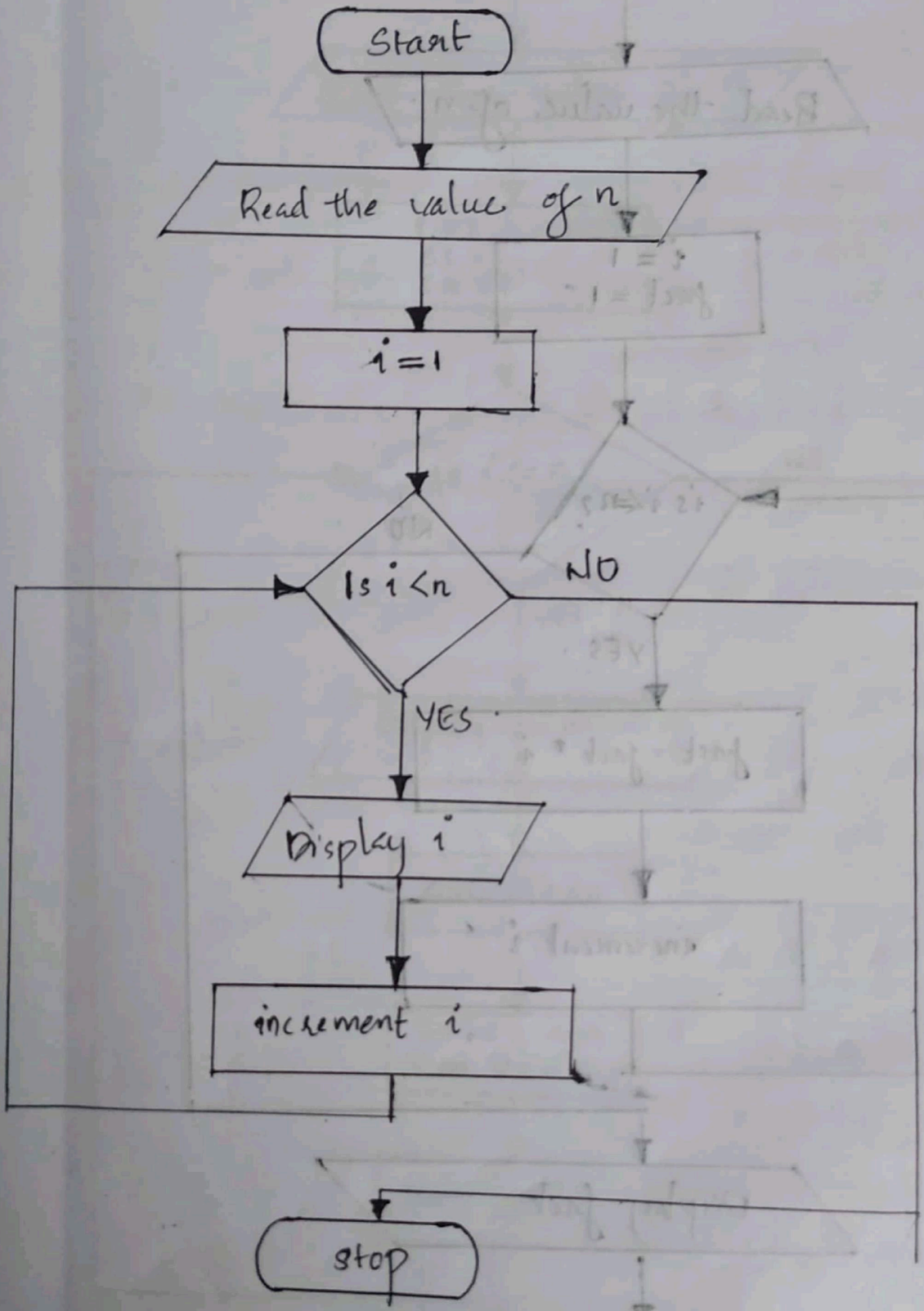
•) Flow chart for finding the roots of a quadratic equation.



⇒LOOPING

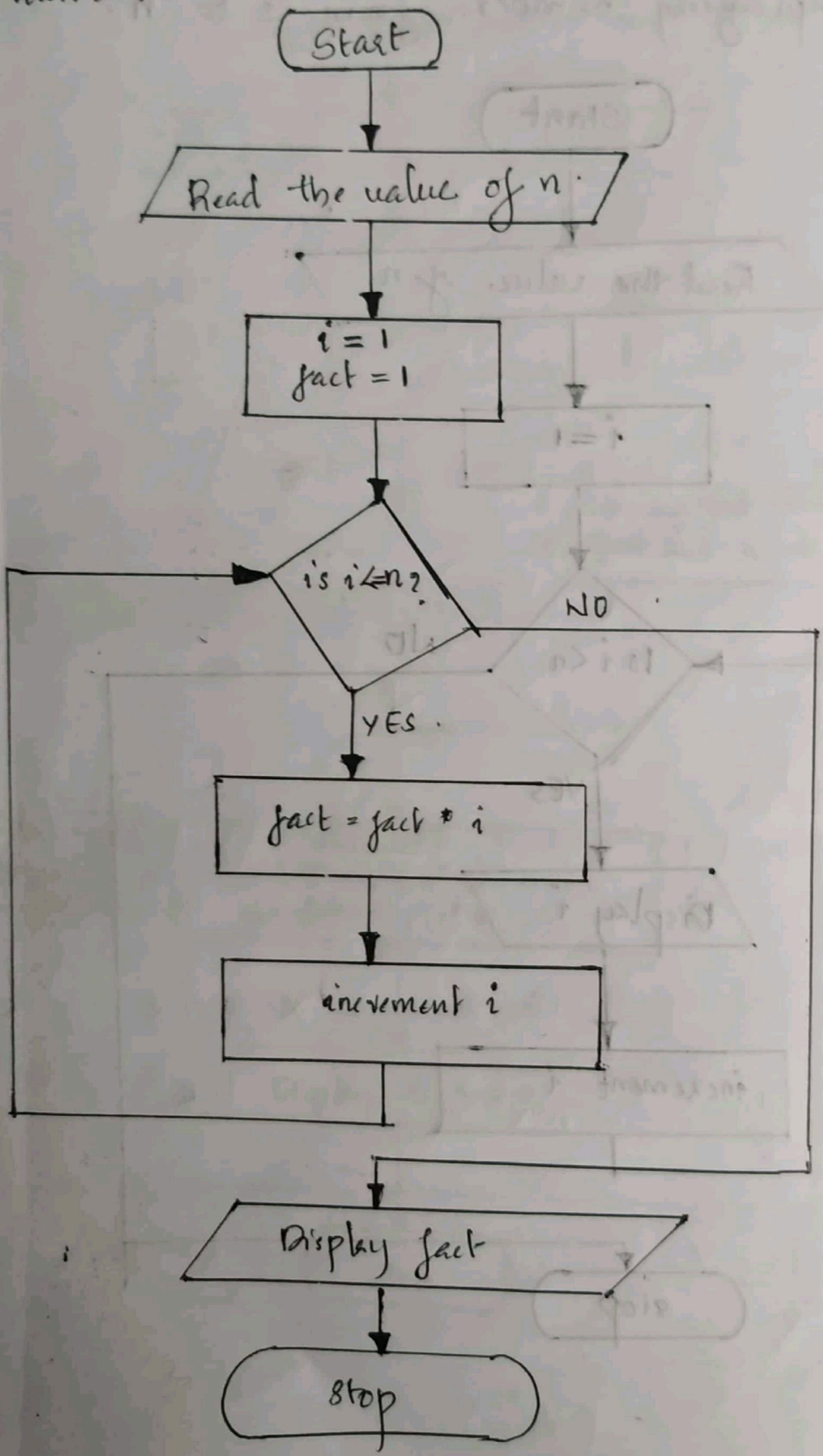
Example 5

• For displaying numbers from 1 to n.



Example 6

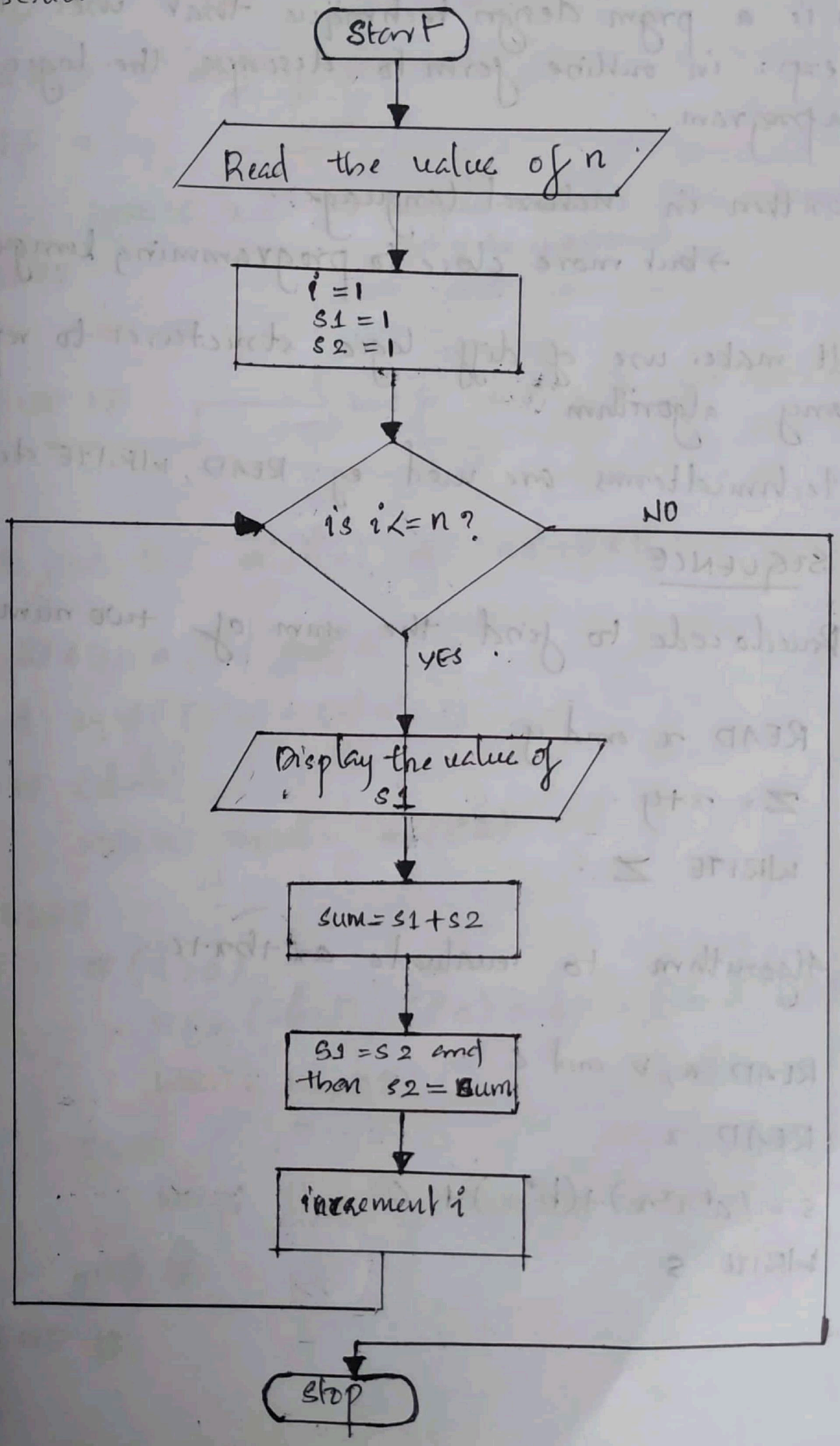
- Flowchart for finding the factorial of a numbers.



H.W

Example:1

Draw the flow chart for displaying fibonacci series.



10/07/2020 PSUEDOCODE

- is a prgm design technique that uses informal exp: in outline form to describe the logic of a program.
- written in natural language.
  → but more close to programming language.
- It makes use of diff: logic structures to represent any algorithm.
- Technical terms are used eg: READ, WRITE etc.

SEQUENCE

1] Pseudo code to find the sum of two numbers.

```
READ x and y.
Z = x+y.
WRITE Z.
```

2] Algorithm to evaluate ax^2+bx+c.

```
READ a, b and c
READ x
s = (a * x * x) + (b * x) + c
WRITE s
```

SELECTION

3] Algorithm to find the largest of two numbers.

```
READ x and y
IF x > y
    WRITE x.        → tabspace indicates, this statement is a
                      part of the if statement.
ELSE
    WRITE y.
END IF              → used for indicating end of a construct.
```

4] To find the solution of ax^2+bx+c.

```
READ a, b and c
d = sqrt[(b*b) - (4*a*c)]
IF (d=0)
    WRITE roots = -b/(2*a).
ELSE
    IF (d>0)
        r1 = (-b+d)/(2*a) and r2 = (-b-d)/(2*a).
        WRITE r1, r2.
    ELSE
        WRITE that the eqn: has imaginary roots.
    END IF
END IF.
```

LOOPING

5] To display numbers from 1 to n.

```
READ n
i=1
IF i<=n DO ——→ represents looping.
    WRITE i
    i++
END DO.
```

6] finding factorial of a number.

* while model →

```
READ n
i=1, f=1
IF i<=n, DO
    f=f*i
    i++
END DO.
WRITE f.
```

OR

```
READ num
f=1
FOR i=1 to n DO
    f=f*i
END DO
WRITE f.
```

HW

7] Write pseudo code for fibonacci series for numbers 1 to

```
READ n.
i=1, S1=1, S2=i
IF i<=n, DO
    WRITE S1
    sum = S1+S2
    S1=S2
```

```
        S2 = sum.
        i++
END DO.
```
