27/01/2023

Thought process to solve a problem

Suppose that we are given 2 number & we have to return their sum. This is an example of problem.

1) Understand the problem. We need to know what is given & what we have to find.
checking the i/p values
2) After knowing the i/p values and also what we need to find, we need to find the approach to solve that particular problem.

Problem → Solution in our mind

At the end we have to make the computer solve the problem. After we have the solution in our mind, just make a rough solution. This rough solution can be in form of flowchart or pseudocode.

Algorithm Steps to solve the problem. This is basically the step-by-step procedure to solve a particular problem.

It is important to note that this rough soln is not understood by the compiter & hence we use a user-friendly language but computer undurstand only binary language i.e machine understandable language.

High level language → Machine language
source code ↳ computer can understand this

Humans will not use the machine language & hence we use high level language which will be converted to machine understandable format. Compiler will do the above conversion.

Using computer to solve a problem

Why to use computer to solve the problem?

Suppose we have to check that 13 is a prime number

$n = 13$ — rem by 1 = 0

$n = 13$ — rem by 13 = 0

13 rem 2 = 1
13 rem 3 = 1
⋮
13 rem 12 = 1

} This is a repetitive task.

The above thing can be manually done for small numbers but it will be a cumbersome task for us to calculate whether a number is prime or not for numbers which are big. Hence this task will be done by the computer.

Note → Problem ⟶ Algorithm ⟶ Soln within 2 min at max.

↓

All game lies here.

Flowchart

This is the graphical representation of an algorithm.

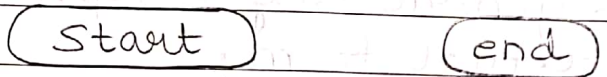
↳ (Series of steps)

Programmers often use it as a program-planning tool to solve a problem. Here we will be using

some symbols.
The process of drawing a flowchart for an algorithm is known as flowcharting.

Components of flowchart

1) Terminator
Indicates the starting and ending state of the algorithm.

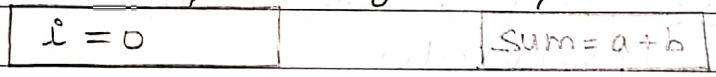


Representation

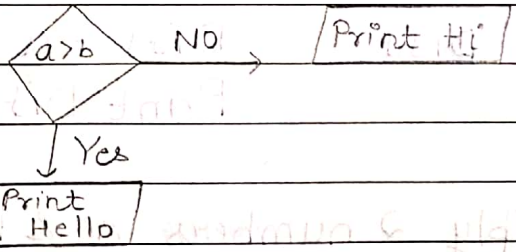
2) Input/output To either take i/p value or to give o/p on the screen

Representation

3) Process block This is used either for initialization or the calculation part ↗ initialization ↗ calculation



4) Decision making block There will be a condition & based on truth value, the direction of algorithm is decided.

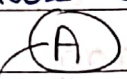


a = 3
b = 5

The o/p will be Hi

5) Arrows These are used to know the flow of the algorithm.

6) Connector This is for function & will be discussed after class of function



↳ A is function name

Pseudocode

Pseudo means false. We don't have to give this code to computer as it won't be able to understand it.

It is generic way of representing algorithm in the textual form.

Ex→ Sum of 2 numbers

```
Start program
Read a & b
sum = a+b
Print sum
End program
```

Note→ There can be multiple ways of writing the pseudocode.

Ex→ Difference of 2 numbers

```
Read a & b
Diff = a-b
Print Diff
```

```
Read a
Read b
Diff = a-b
Print Diff
```

```
Read a
Read b
Print a-b
```

Ex→ Multiply 2 numbers a & b

```
Read a & b
Prod = a * b
Print Prod
```

Ex→ Print average of 2 numbers

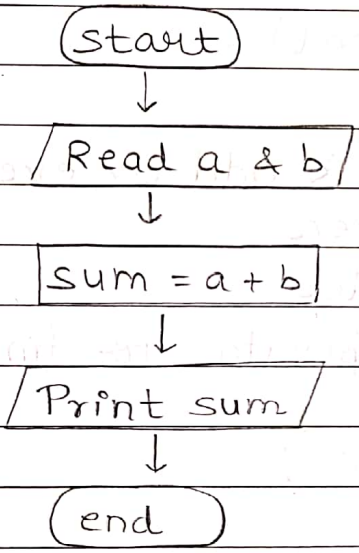
Read a & b

$avg = \frac{(a+b)}{2}$

Print avg

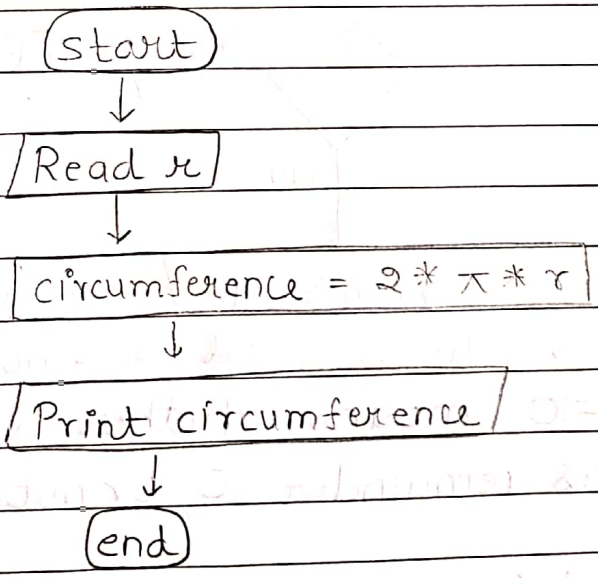
Practice questions

1) Add 2 numbers by taking input



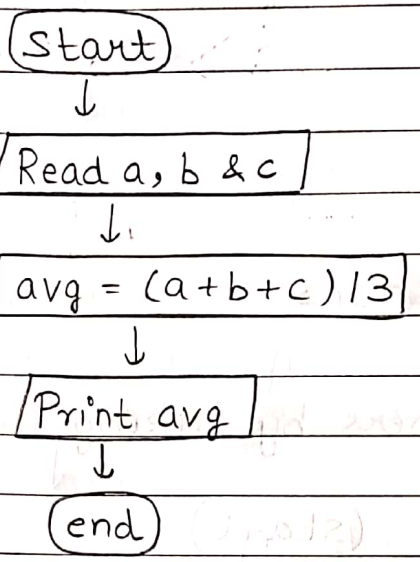
Read a & b
sum = a+b
Print sum

2) Find circumference of circle



Read r
cir = 2*3.14*r
Print cir

3) Average of 3 numbers



Read a, b & c

$avg = \frac{(a+b+c)}{3}$

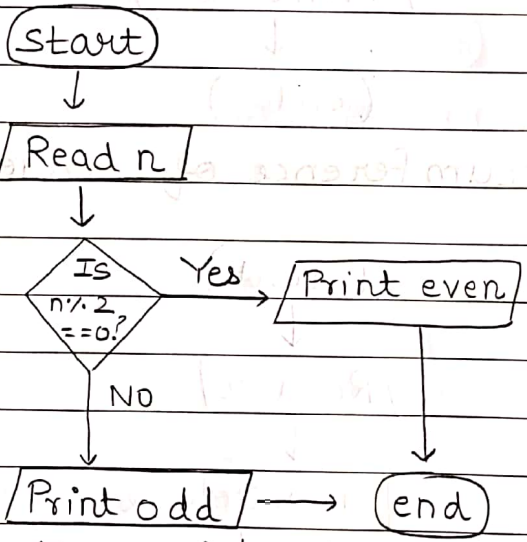
Print avg

4) Check if number is odd or even

n rem 2 = 0 → even

n rem 2 ≠ 0 → odd

Here we use % operator to find the remainder.

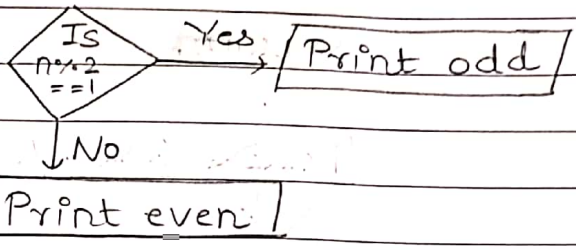


→ indicates equal to & not assignment operator

n%.2 (==) 0 means whether when n divided by 2 gives remainder 0 or not.

% → modulo operator

Alternative



as n %. 2 == 1 , this means odd number.
7 %. 2 = 1 hence 7 is odd number.

Pseudocode

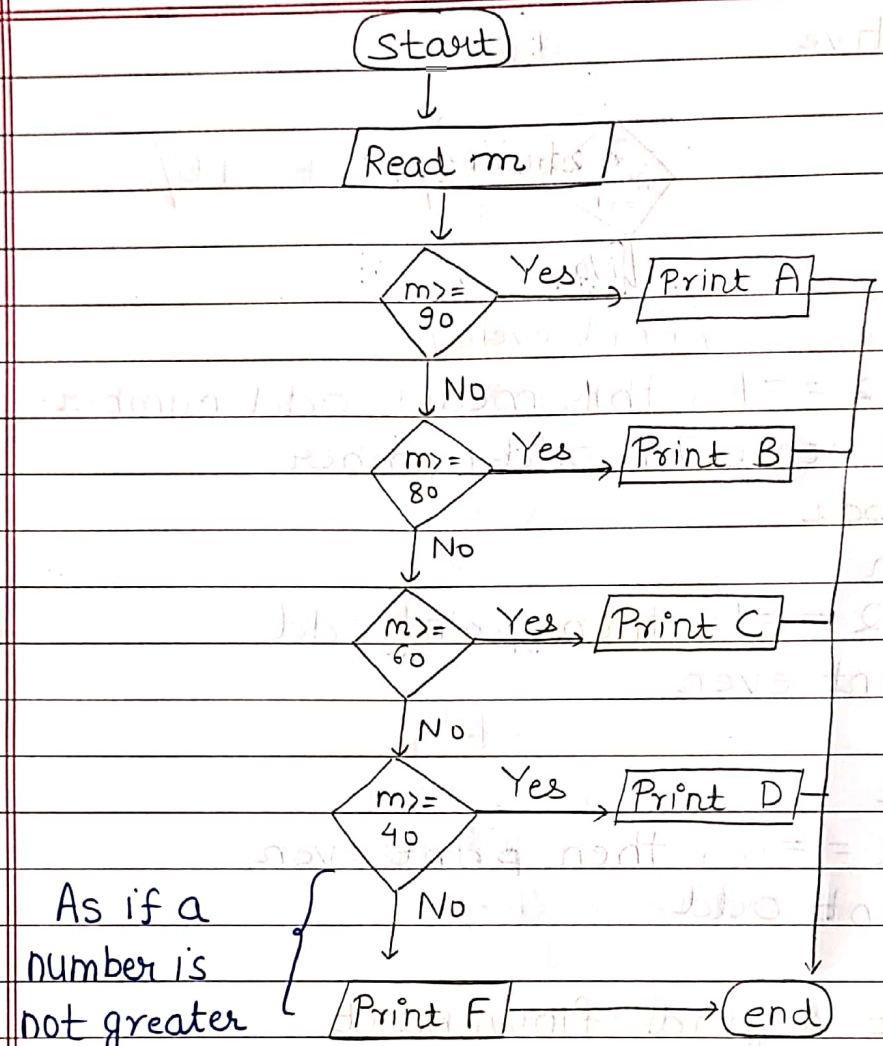
* Read n
if n %. 2 == 1 , then print odd
else print even

* Read n
if n %. 2 == 0 , then print even
else print odd

5) Student & grade flowchart

marks >= 90 → A
marks >= 80 → B
marks >= 60 → C
marks >= 40 → D
marks < 40 → F

Pseudocode
Read marks
if marks >= 90 , print A
else if marks >= 80 , print B
else if marks >= 60 , print C
else if marks >= 40 , print D
else if marks < 40 , print F ; else print F

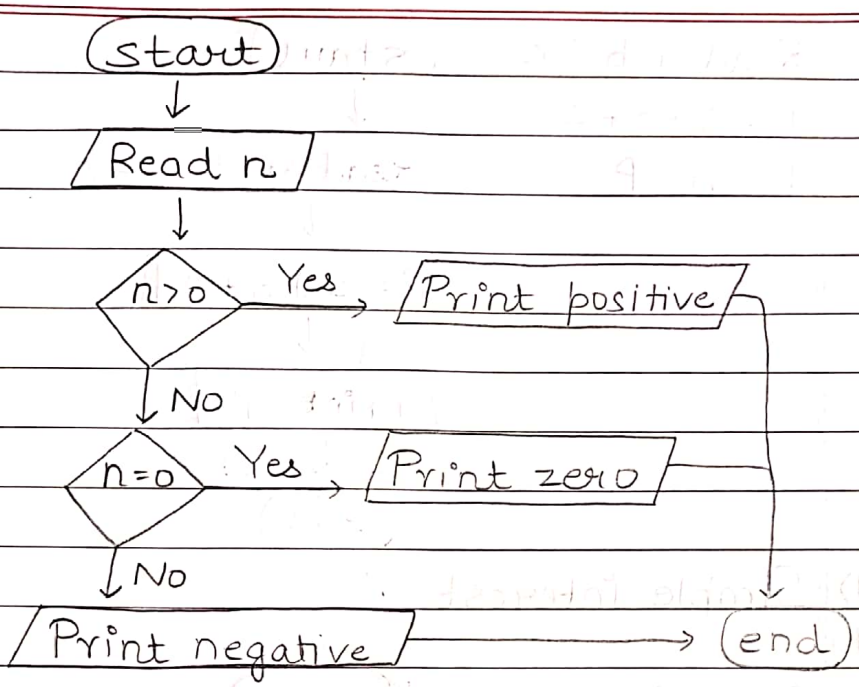


As if a number is not greater & also not equal, this means that number will be obviously lesser than 40.

6) Check number is +ve, -ve or 0

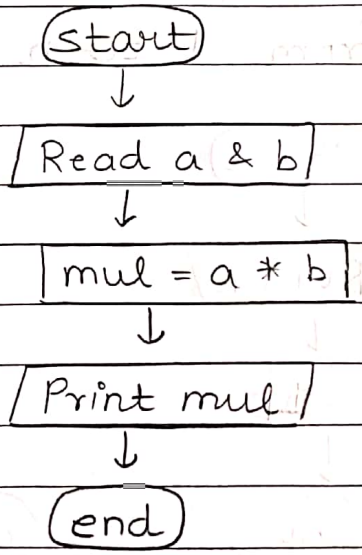
Read n
if n>0 print positive
else if n=0 print zero
else print negative

no need to check as if it is not positive & also not zero, then it will be negative only.



Dry run
This is process of taking any random value & then moving in the flowchart to check whether our algorithm is running fine or not.

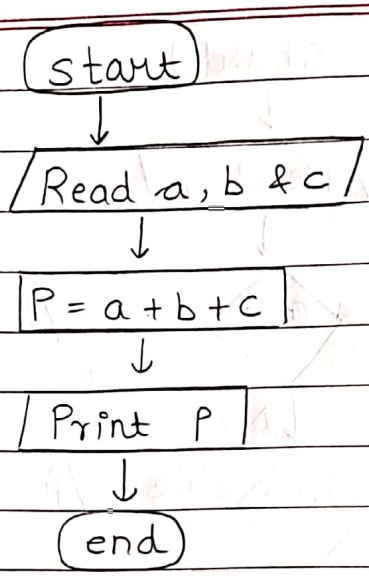
7) Multiply 2 number by taking input
(HW)



Read a & b
mul = a * b
Print mul

8) Find perimeter of triangle
(HW)

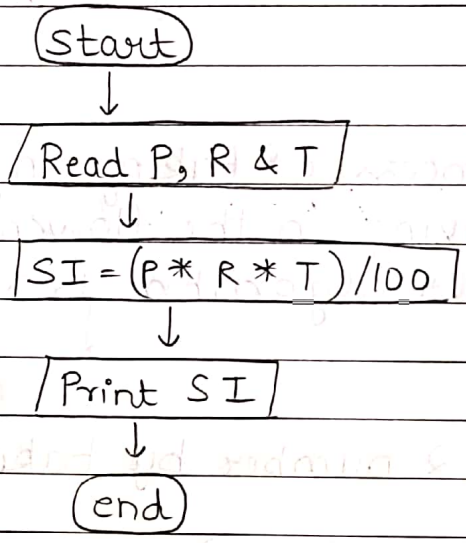
Read a, b & c
P = a+b+c
Print P



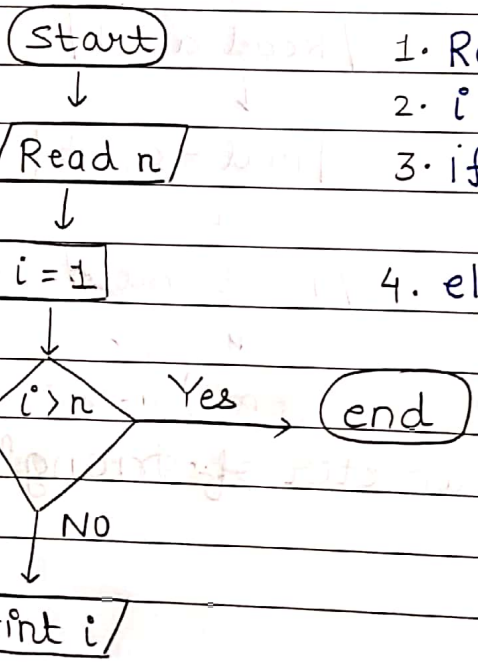
9) Simple interest

HW

Read P, R & T
$SI = \frac{(P * R * T)}{100}$
Print SI



10) Print counting from 1 to n



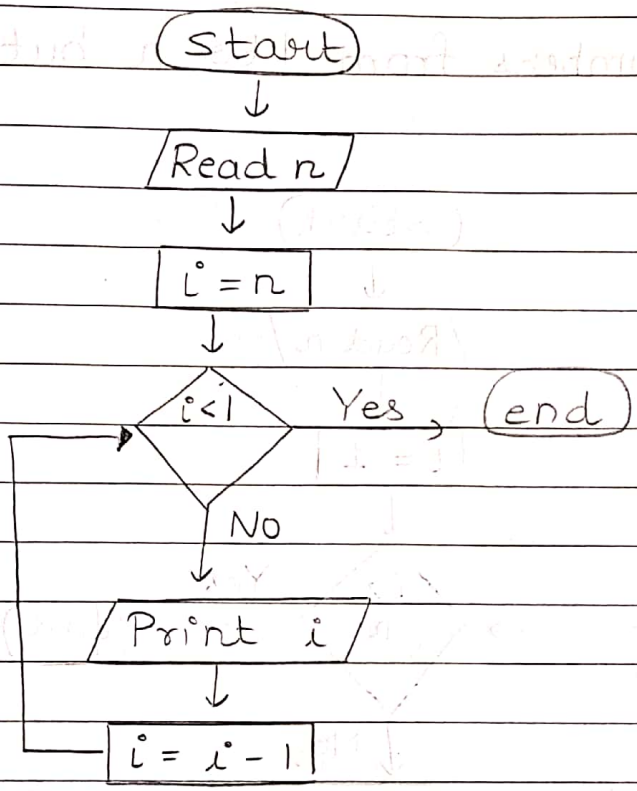
1. Read n
2. i = 1
3. if i > n
exit
4. else print i
i = i+1
go to 3

Looping concept

Here we have used the concept of looping.

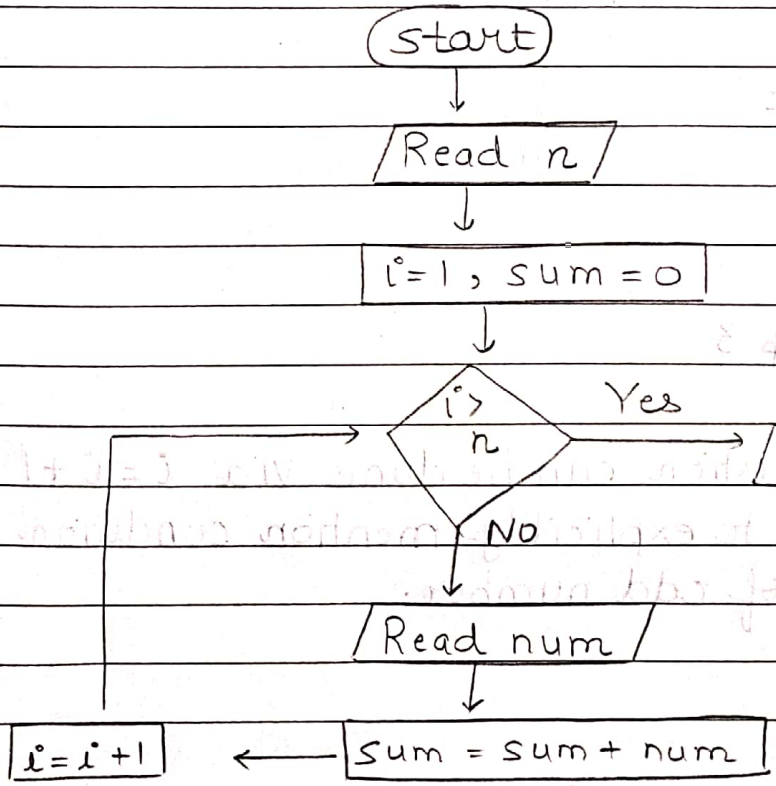
11) Print counting from n to 1

(HW)



1. Read n
2. i = n
3. if i<1
   exit
4. else print i
   i = i - 1
   goto step 3

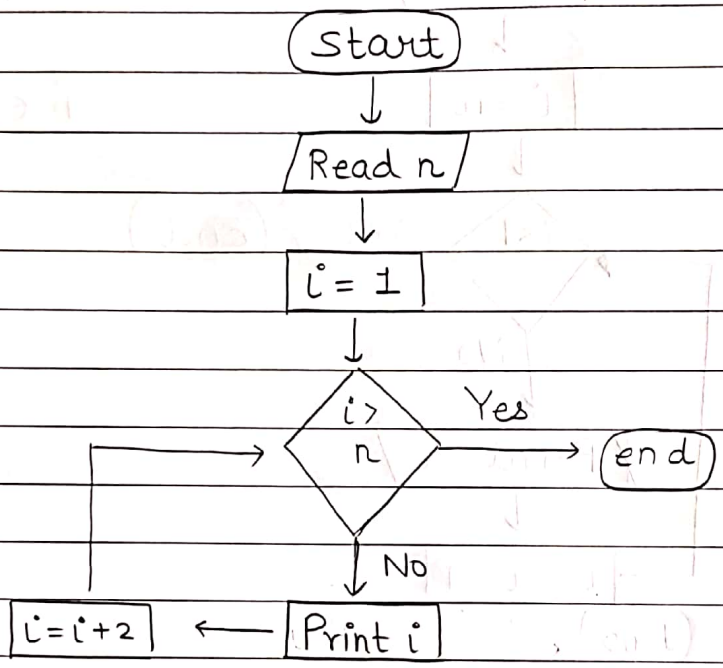
12) Add n numbers from user



1. Read n
2. i=1, sum=0
3. if i > n
   print sum
   exit
4. else read num
   sum = sum + num
   i = i + 1
   goto step 3

The question means that if n=3, then we have to 3 numbers as i/p & then print sum of these 3 numbers.

13) Print numbers from 1 to n but only odd numbers



1. Read n
2. i = 1
3. if i > n, exit
4. else
   print i
   i = i + 2
   goto step 3

Note→ The above question can be done via i = i + 1 also but we have to explicitly mention condition for checking of odd number.