UNIT-1

Components of a computer system

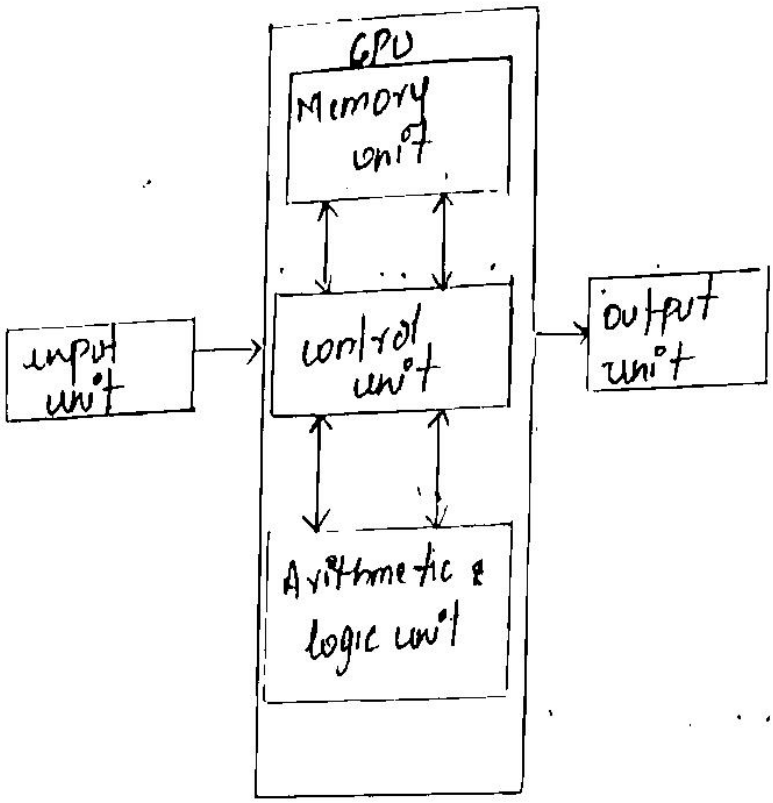
A computer is any machine that can be programmed to carry out a set of algorithms and arithmetic instructions

A computer is a (or) machine that can be instructed to carry out sequences of arithmetic or logical operations automatically via computer programming. Modern computers have the ability to follow generalized set of operations, called programs.

Components of a Computer

All types of computers follow the same basic logical structure and perform the basic operations for converting raw data into information useful to their users.

- Motherboard
- Central processing unit (CPU)
- Graphics processing unit (GPU) also known as video card.
- Random Access memory (RAM), also known as volatile memory.
- Storage: solid state drive (SSD) or Hard disk drive (HDD)



Motherboard

Motherboard acts as a backbone of any technological device and it holds all the major components of a computer including the hard drive, processor, memory and peripheral ports like the USB.

Most motherboards are expandable i.e we can replace components as long as they are compatible.

Graphic card (or) video card

A graphics (or) video card can come in two varieties - Integrated (or) expansion.

An integrated video card is connected directly to the motherboard and is usually found as part of the processor.

An expansion video card is a separate card that is connected to another part of the motherboard called an expansion port.

The job of the video card is to create the graphics and images that can be shown on a monitor.

Processor

The processor - also known as central Processing unit (CPU) is the brain of the computer.
CPU performs all types of data processing operations
It stores data, intermediate results and instructions
It controls the operation of all parts of the computer.

CPU itself has the following three components

- ALU (Arithmetic logic unit)
- Memory unit
- control unit.

Arithmetic Logic unit

The unit consists of two subsections namely

- Arithmetic section: is to perform arithmetic operations like addition, subtraction, multiplication and division.

Logic Section: Function of logic section is to perform logic operations such as comparing, selecting, matching and merging of data.

Memory (or) storage unit

This unit can store instructions, data and intermediate results. This unit supplies information to other units of the computer when needed. It is also known as internal storage unit or the main memory or the primary storage (or) Random Access Memory (RAM)

Its size effects speed, power and capability.

Primary and secondary memory are two types of memories in the computer

Functions of the memory unit are

- It stores all the data and the instructions required for processing
- It stores result intermediate results for processing
- It stores the final results of processing before these results are released to an output device
- All input and outputs are transmitted through main memory

Control Unit

This unit controls the operations of all parts of the computer but does not carry out any actual data processing operations

Functions of the unit are

- It is responsible for controlling the transfer of data and instructions among other units of a computer
- It manages and coordinates all the units of a computer
- It obtains the instructions from the memory, interprets them, and directs the operation of a computer
- It communicates with input/output devices for transfer of data or results from storage
- It does not process (or) store data

Random Access Memory (RAM)

It is the internal memory of the CPU for storing data, program and prog result. It is a read/write memory which stores data until the ~~system~~ machine is working.

Access time in RAM is independent of the address, that is each storage location inside the memory is as easy to reach as other locations and takes the same amount of time.

Data in the RAM can be accessed randomly but it is very expensive.

Data is volatile ie data is erased or lost when we switch off the computer or on power lost.

RAM is of two types
- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Hard drive : Hard drive is where programs and files are stored. HDD are comprised of a series of magnetized disks which store the data. These disk spin under a magnetic arm which can read and write data.

Newer hard drives are called solid-state drives (SDD) and use electrical circuits to store data. These are much faster than HDDs

Static RAM - is one which requires the constant flow of the power to retain data inside it.

Dynamic RAM - needs to be refreshed to retain the data it holds - capacitors (capacitors, transistors)

ROM (Read only Memory)

- Rom - is non-volatile memory
- the capacity of ROM is comparitively smaller than RAM
- It is slower and cheaper than RAM
- the data in ROM can only be read by CPU
- ROM stores the instruction that computer requires during Bootstrapping.

Types of ROM

PROM - Programmable ROM - it can be modified only once by user

EPROM - electrically erasable and PROM - the content of the ROM can be erased using ultraviolet rays and the ROM can be programmed

EEPROM - electrically erasable PROM, it can be erased electrically and reprogrammed about ten thousand times

**SSD are faster than HDD

Introduction to Flowchart and Algorithms.

A set of steps involving arithmetic and computation and/or logical manipulation is required to obtain the desired result for a given problem. The desired result can be achieved through different courses of action and a variety of paths, and this is called "law of equifinality"

The most effective way of achieving a solution is called the optimum way. The optimum way of solving a problem to get the desired result can be achieved by analysing different strategies for the solution and then selecting the way that can yield the result in least time using the minimum amount of resources.

A type of analysis called task analysis is required to reach the solution from a problem definition that states what is to be achieved.

A set of steps that generates a finite sequence of elementary computational operations leading to the solution of a given problem is called an Algorithm.

Pictorial representation can be used as a substitute for an algorithm to understand more clearly and that is called as Flowchart

Flowchart is a diagramatic representation of an algorithm, boxes of different shapes are used to denote different types of operations and these boxes are connected by lines with arrows denoting the flow (or) direction. The connecting lines are known as flow lines.

Flowcharts may be classified into two categories.

i) Program flowchart

Prog flowchart act like mirrors of computer progs in terms of flowcharting symbols. They contain the steps of solving a problem unit for a specific result

ii) System Flowchart

Contain the solution of many problem units together that are closely related to each other and interact with each other to achieve goal.

Program flowchart

- Is an extremely useful tool in prog development
- Program flowchart is a pictorial representation of the logic of a program due to which it is easy to detect any error (or) omission from the progflowchart than from a program.
- Program flowchart can be followed easily and quickly.
- The type of documentation which may be of great help if the need for program modification arises in future.

The following five rules should be followed while creating program flowcharts.

1) only the standard symbols should be used
2) the program logic should depict the flow from top to bottom and from left to right

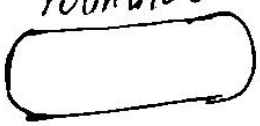
3) Each symbol used in a program flowchart should contain only one entry point and one exit point with the exception of decision symbol. This is known as single rule

4) The operations shown within a symbol of a program flowchart should be expressed independently of any programming language

5) All decision branches should be well-labeled.

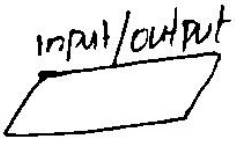
The following are the standard symbols used in program flowcharts.

Terminal



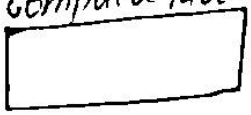
used to show the beginning and ending of a set computer-related processes

Input/output



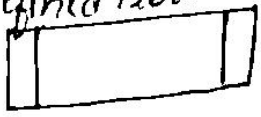
used to show any input/output operation

Computer Processing



used to show any processing performed by a computer system

Predefined Processing

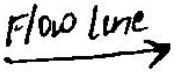


used to indicate any process not specially defined in the flowchart

Comment

used to write any explanation

Flow line

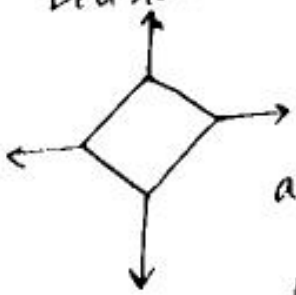


used to connect the symbols

Document I/O

used when input comes from the document and output goes to the document

Decision



used to show any point in the process where a decision must be made to determine further action

on-page connector

used to connect parts of a flowchart continued on the same page

off-page connector

used to connect parts of a flowchart continued to separate pages

⇒ Flowcharts can be used to show the sequence of steps for doing any job.

⇒ A set of simple steps involving accepting inputs, performing arithmetic operation on the inputs, and showing them to the users demonstrate the sequence logic structure of a program

~~SSD are faster~~

Flowchart

① The algorithm for the flowchart about cooking

step 1: Take the rice to be cooked

2: Procure the container

3: Procure the water

4: wash the rice in the water

5: Put the rice into the container

6: Pour water into the container

7: If water level = 1 inch above the rice

Then goto step 8

Else Goto step 6

endif

8: Light the burner of the stove

9: If the rice is boiled

Then goto step 12

else goto step 10

endif

10: Heat the container

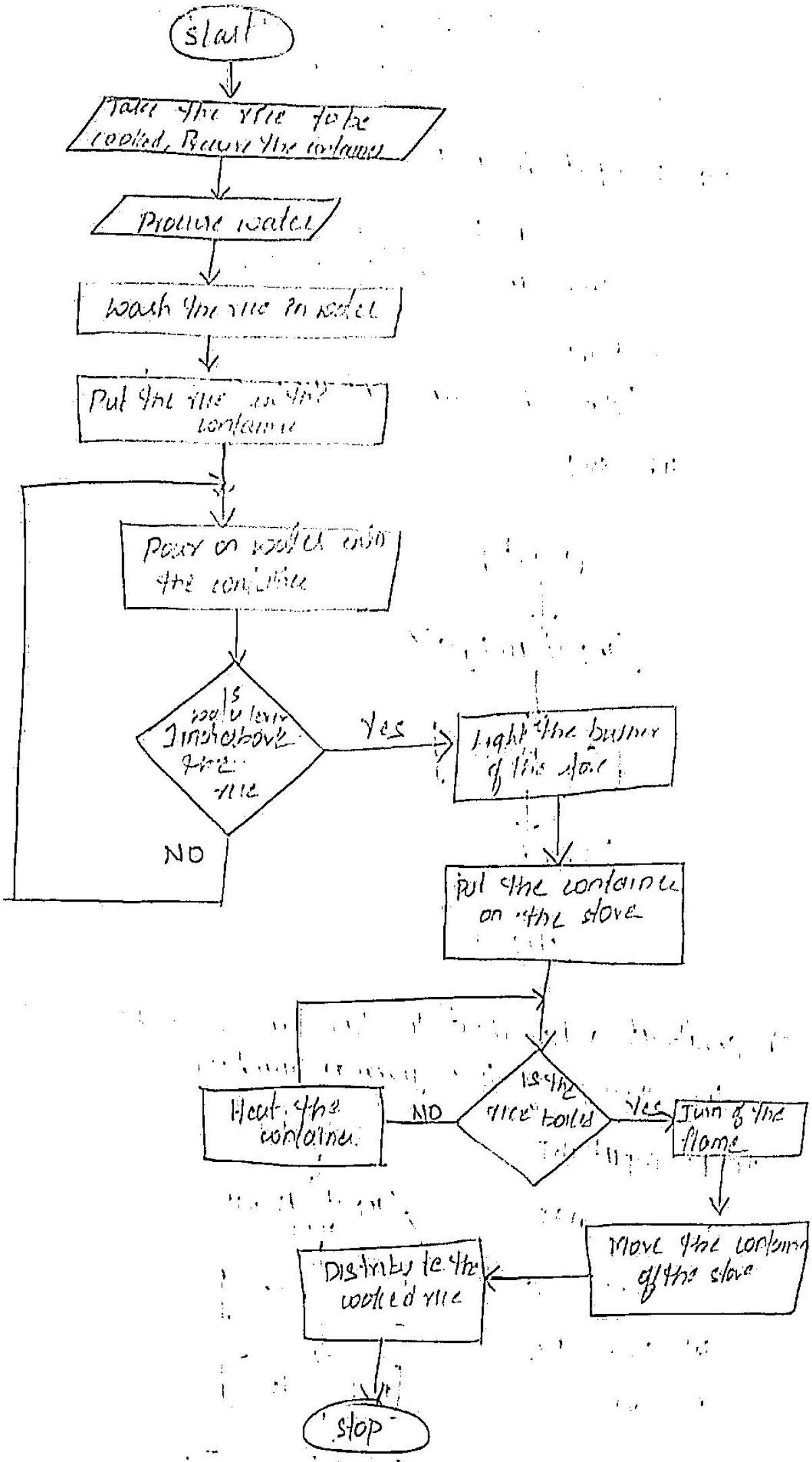
11: go to step 9

12: Turn off the flame

13: Move the container off the stove

14: Distribute the cooked rice

15: stop


Start
Take the rice to be cooked, Remove the container
Pressure water
Wash the rice in water
Put the rice in the container
Pour on water into the container
Is water level 3inch above the rice
Yes
NO
Light the burner of the stove
Put the container on the stove
Heat the container
NO
Is the rice boiled
Yes
Turn off the flame
Move the container off the stove
Distribute the cooked rice
Stop

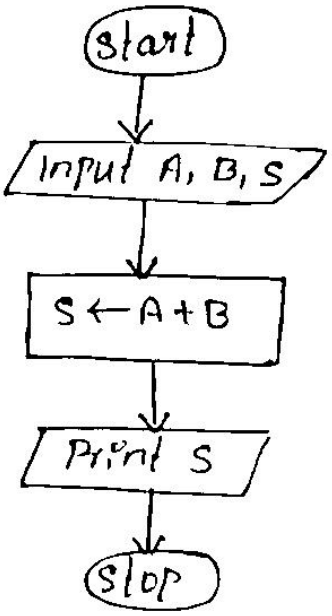
② Draw a flowchart to show how the sum of two numbers can be obtained.

step 1: Input to A, B

2: S ← A+B
(Store the sum of A & B)

3: Print S
(Show the sum obtained in step 2)

4: stop



③ Construct a flowchart to show the procedure for obtaining the average of two given numbers.

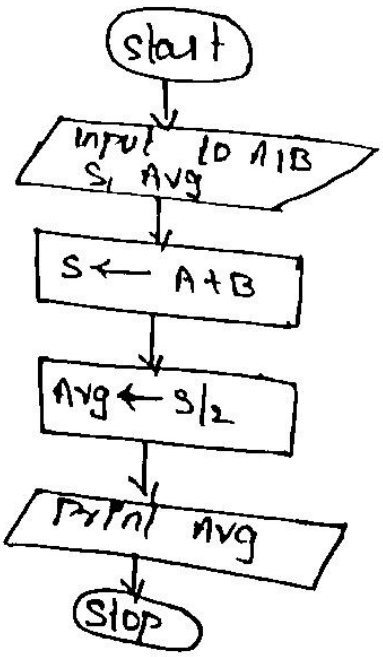
step 1: input A, B

2: S ← A+B

3: Avg ← S/2

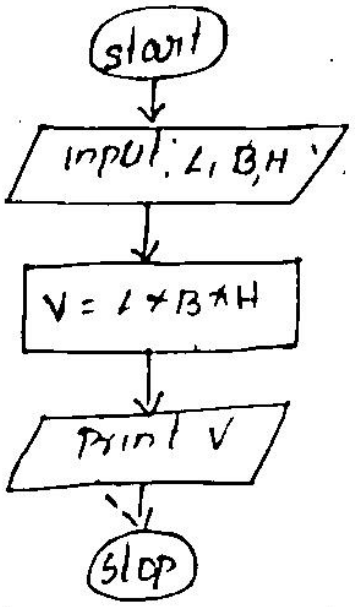
4: Print Avg

5: stop



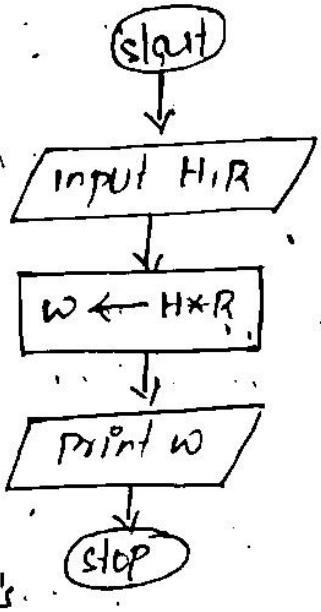
④ Construct a flowchart to show the how to obtain the volume of a rectangular box

step 1: input to L, B, H
2: compute V = L * B * H
3: Print V
4: stop



⑤ Construct a flowchart to show how to obtain the daily wage of a worker on the basis of the hours worked during the day

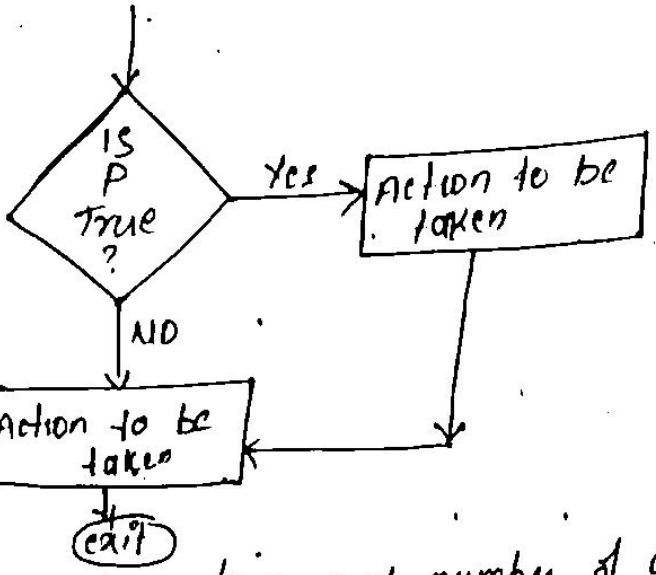
step 1: input to H, R
2: compute W ← H * R
3: Print W
4: stop



6) Find the product of two no's
7) Find the remainder when one number is divided by the other
8) Find the area of a parallelogram
9) Find the area of a four walls of a rectangular room
10) Find the area and perimeter of a circular plot
11) Find the area and perimeter of a square

* Decision-making

- The process of decision-making is implemented through a logic structure called <u>Selection</u>
- Here a predicate, also called a condition, is tested to see if it is true (or) false.



- A flowchart may contain any number of decision boxes depending on the processing requirements, and the boxes may appear in any sequence depending on the program logic decided.

⇒ Develop a flowchart to show how the profit (or) loss for a sale can be obtained?

<u>Algo</u> step 1: start

2: input to CP, SP

3: If ~~CP < SP~~ SP > CP 50 < 40

then goto step 4

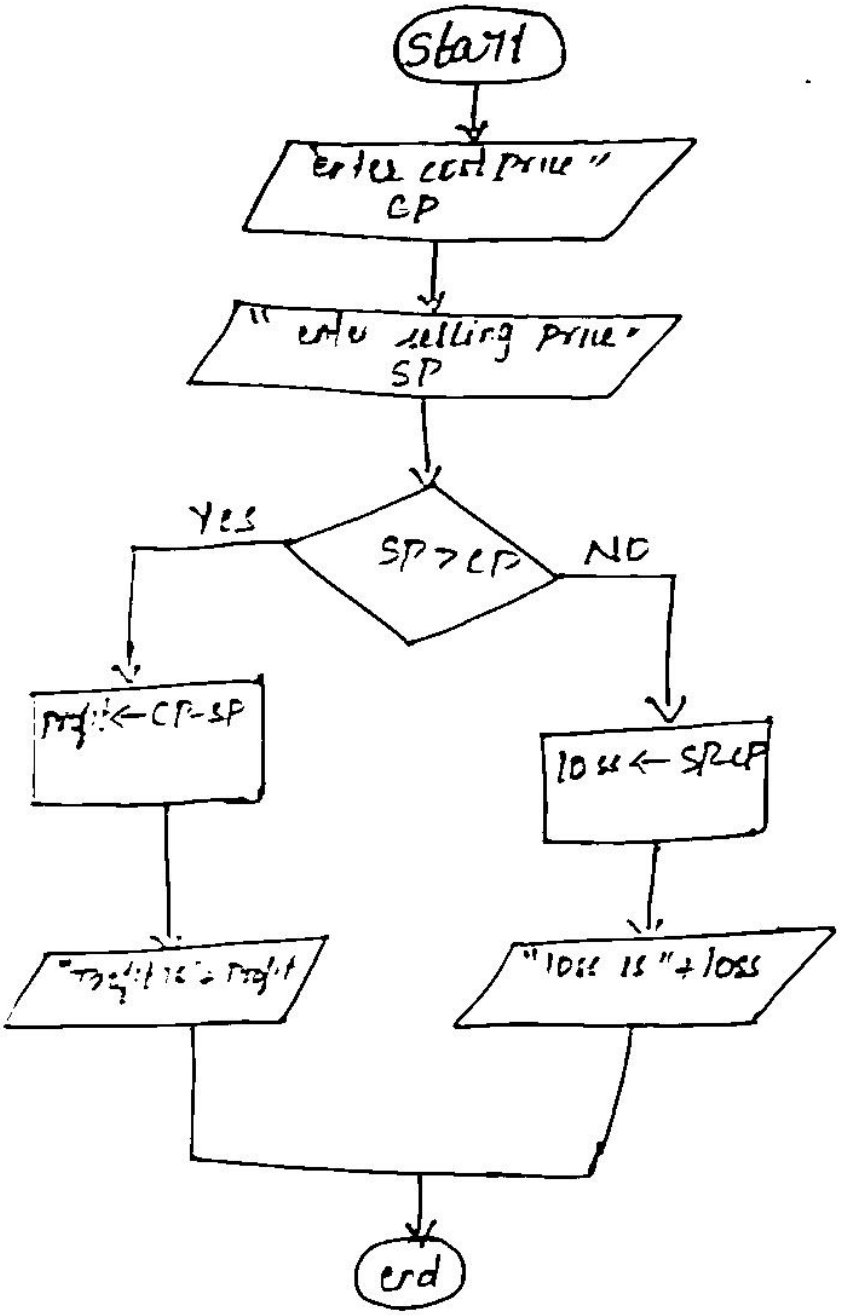
else goto step 5

end if

4: loss = SP - CP

5: profit = CP - SP

end



=, write an algo and draw flowchart to find the greatest of 3 no's

step 1: start

2: input to A, B, C

3 if A > B and A > C

print A

else go to step 4

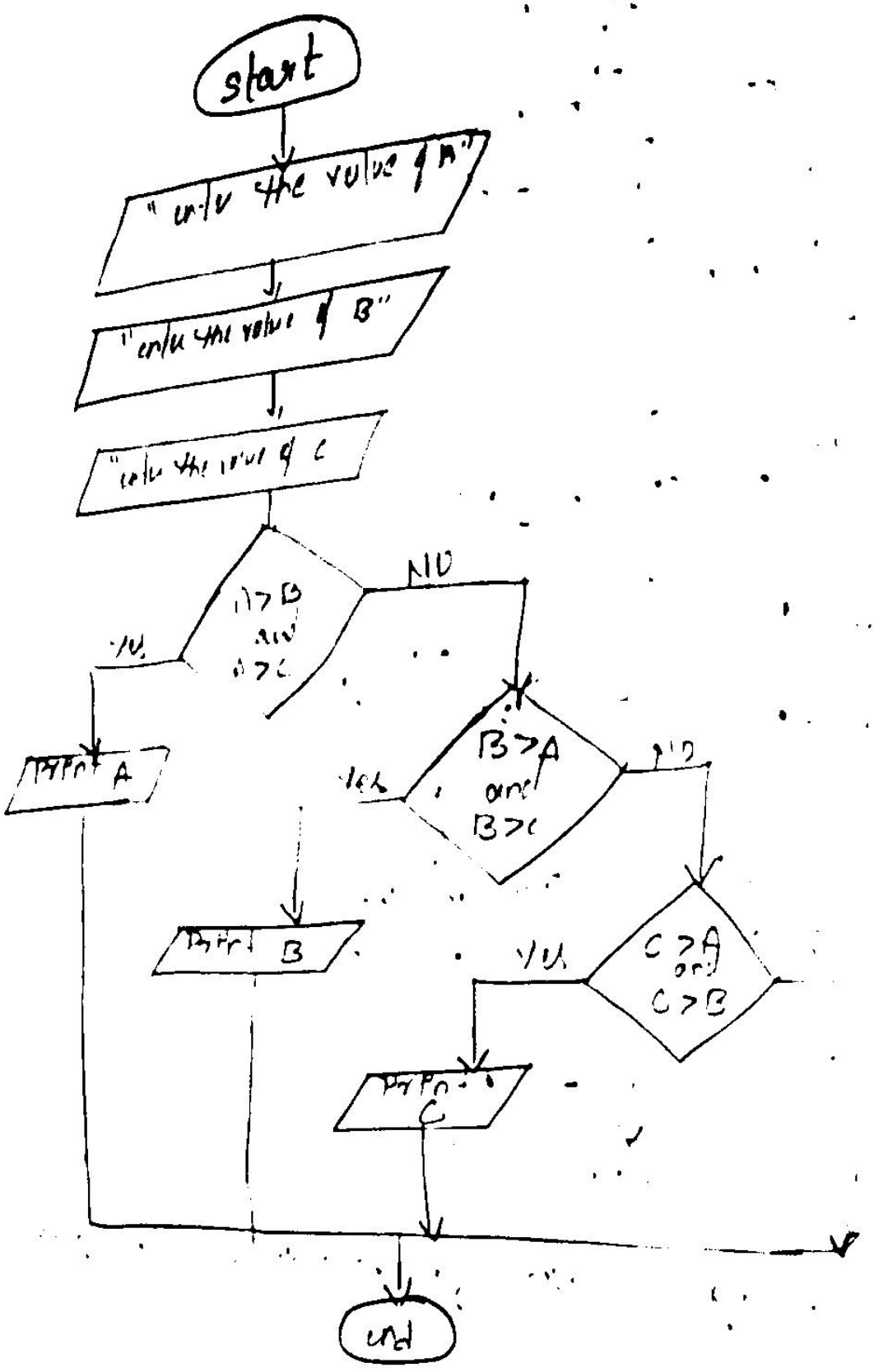
4 B > A and B > C

print B

else go to step 5

5 if C > A and C > B

print C. exit



⇨ write the algo and draw flowchart to find the given the character is vowel or not

Fundamental Algorithms

Exchanging the values of two variables

- Interchanging the values associated with two variables involves fundamental mechanism that occurs in sorting and data manipulation algorithms.

suppose the variables a and b are assigned values as

a	b
721	463

to exchange values of both the variables with the following assignments

① a := b → ~~a := b~~ → 463 = b

② b := a → 463 = a ✗

the first assignment works fine and exchanged the value in variable 'a' but has left the value of 'b' untouched.

To overcome this type of problem, need to introduce a temporary variable 't' and copy the original value of 'a' into this variable before executing step ①

```
t := a   — ③
a := b   — ①
b := t   — ②
```

t	a	b
721	463	721

Algorithm description

1. Save the original value of a in t
2. Assign to a the original value of b
3. Assign to b the original value of a stored in t

Pascal implementation

```
Procedure exchange (var a, b: integer)
    var t: integer
begin {save the original value of a then exchange a and b
        { assert a = a0 ∧ b = b0 }
    t := a
    a := b
    b := t
end.
```

uses : The use of an intermediate temporary variable allows exchange of two variables to proceed correctly.

Application sorting algo.

(2) Given two variables of integer type a and b, exchange their values without using a third temporary variable.

sol

a	b
50	30

To exchange the values of both the variables with the following assignments

step 1- a := b → a := 30

2- b := a → b := 30

where the step ① is successful in exchanging the value of variable a but failed in step ②

To exchange the values of variables without using third variable, can be done by using arithmetic operations.

i.e

step			
~~1~~	~~b := b a~~	~~→ b :=~~	
3	a := a+b	50 + 30	= 80
4	b := a-b	80 - 30	= 50
5	a := a-b	80 - 50	= 30

a	b
30	50

Algorithm description

1) add the values of both variable and save it in a
2. subtract the b value with a new value and assign to b
3. subtract the new value of b with new value of a and assign it to b.

Counting

- counting mechanisms are very useful in computer algo
- count must be made of the number of items in a set which possess some particular property or which satisfy some particular condition or conditions

Suppose we are given the set of marks

55, 42, 77, 63, 29, 57, 89

with the condition ≥50

To make a count of this marks, let us start from left

first mark 55 ≥ 50 → ✓ 1 student mark is counted

Marks	counting details for passes		
	Pres	current	
55	0	1	≥50 ✓
42	1	1	≥50 X
77	1	2	≥50 ✓
63	2	3	≥50 ✓
29	3	3	≥50 X
57	3	4	≥50 ✓
89	4	5	≥50 ✓

The current count reflects the total NO. of students passed the exam

To calculate the current count and previous_count

to current_count = Previous_count + 1 —(1)

Previous_count = current_count —(2)

these two steps must be repeatedly applied to obtain the count required.

The step(1) can also written as

current_count = current_count + 1
(New) (old)

as old current_count = = previous count.

(or)

new current count is obtained by adding 1 to the old value of the current_count and in such a case the existence of previous_count is unnecessary.

The essential steps in our pass-counting algorithm be written as

while less than n marks have been examined do

(a) Read next mark

(b) if current mark satisfies pass requirement then add one to count.

Algorithm description

1) Prompt then read the number of marks to be processed
2 Initialize count to zero
3 while there are still marks to be processed repeatedly do
(a) read next mark
(b) if it is a pass (ie ≥ 50) then add one to count
4 write total number of passes.

Pascal implementation.

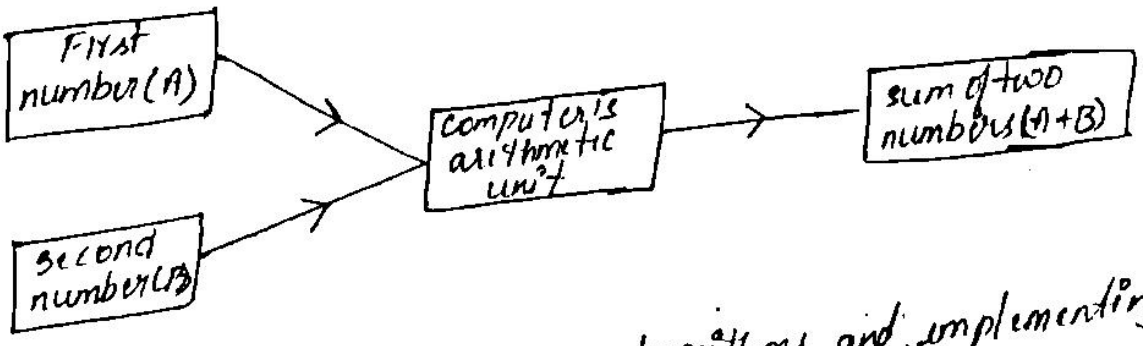
```
Program passcount (input, output)
  const passmark = 50;
  var count {contains number of passes on termination}
      i {current number of marks processed}
      m {current mark}
      n {total number of marks to be processed initially}

begin {count the number of passes (>= 50) in a set of marks}
  writeln('enter a number of marks n on a separate
                    line followed by the marks');
  readln(n);
  {assert: n >= 0}
    count := 0;
    i := 0;
{invariant: count = number of marks in the first i
      read that are >= passmark ∧ i <= n}
    while i < n do
begin {read next mark, test it for pass and update
                    count, if necessary}
      i := i + 1;
      read(m);
    if eoln(input) then readln;
    if m >= passmark then count := count + 1
    end;
{assert: count = number of passes in the set of n marks read}
  writeln('number of passes = ', count)
end.
```

- Initially, and each time through the loop, the variable count represents the number of passes.
- On termination, count represents the total number of passes in the set.
- Because i is incremented by 1 with each iteration, eventually the condition i <= n will be violated and the loop will terminate.

Applications :- All forms of counting

Summation of a Set of Numbers

The computer has a built-in device which accepts two numbers to be added, performs the addition, and returns sum of the two numbers.



- A fundamental goal in designing algorithms and implementing Progs is to make the programs general enough to handle a wide variety of input conditions.

$$S = (a_1 + a_2 + a_3 + \cdots + a_n)$$

or $S = \sum_{i=1}^{n} a_i$ to add n numbers which is not practical

The core of the algorithm for summing n numbers involves a special step followed by a set of n iterative steps

that is,

1. compute first sum (s = 0) as special case
2. Build each of the n remaining sums from its predecessor by an iterative process
3. write out the sum of n numbers

Algorithm description

1. Prompt and read in the number of numbers to be summed
2. Initialize sum for zero numbers
3. while less than n numbers have been summed repeatedly do
   (a) read in next number,
   (b) compute current sum by adding the number read to the most recent sum
4. write out sum of n numbers.

Pascal implementation

```
Program sum(input, output);
  var i{summing loop index}:
      n{number of numbers to be summed}: integer;
      a{current number to be summed};
      s{sum of n numbers on termination}: real
        begin{computes sum of n real numbers for
              n>=0}
```

```
writeln{'input n on a separate line followed by the
                    numbers to be summed'};
readln(n);
  p:=0;
  s:=0.0;
{invariant: s = sum of first p numbers read \in n}
  while i<n do
    begin {calculate successive partial sums}
      i:=i+1;
      read(a);
      if eoln(input) then readln;
        s:=s+a;
      end;
      writeln{'sum of n= ', n, 'numbers =', s}
end.
```

Notes on design

1. To sum n numbers (n-1) additions must be performed.
2. Initially and each time through the loop, the sum s reflects the sum of first i numbers read.
3. On termination (i=n) s will represent the sum of 'n' numbers.

Applications

Average calculations, variance and least square calculations

2 3.1 Design an algo to compute the average of n numbers.

2 3.2 Redesign the algorithm so that it only needs to perform n-1 additions to sum n numbers

2 3-3 Generate the first n terms of the sequence

1 2 4 8 16 32

Factorial Computation

Given a number n, compute n factorial (written as $n!$) where $n \geq 0$

Algorithm Development

Start the development of algo by examining of $n!$

$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$ ~~for~~ for $n \geq 1$

and by the rule

$0! = 1$

$n!$ can always be obtained from $(n-1)!$ by simply multiplying it by n (for $n \geq 1$) that is

$n! = n \times (n-1)!$ for $n \geq 1$

let say $P = 0 = 1!$ we can describe the steps as

P := 1 —(1) $= 0!$

P! = P*1 } →(2...n+1) $= 1!$

P := P*2 } $= 2!$

P := P*3 } $= 3!$

For the general $(i+1)^{th}$ step we have:

$P_i := P * i$

- This general step can be placed in a loop to iteratively generate $n!$.
- This step allows us to take advantage of the fact that the computer's arithmetic unit can only multiply two numbers at a time.
- The central part of the algo for computing $n!$ involves special initial step followed by 'n' iterative steps.
  1. Treate $0!$ as a special case ($P := 1$)
  2. Build each of the n remaining products P from its predecessor by an iterative process
  3. write out the value of 'n' factorial.

Algo description

1. Establish n, the factorial required where $n \geq 0$
2. Set product P for $0!$. Also set product count to. 3210..
3. while less than 'n' products have been calculated repeatedly do
   (a) increment product count
   (b) compute the i^{th} product 'p' by multiplying 'i' by the most recent product.
4. Return the result $n!$

Pascal implementation

```
function nfactorial(n: integer): integer;
var i{loop index representing ith factorial}: integer;
  factor{i!}: integer;
begin {computes and returns n! for n>=0}
  {assert: n>=0}
  factor:=1;
  {for i:= to n do
    factor:= i* factor;
    n.factorial := factor
  end
```

Notes

This algorithm uses n multiplications to compute n!

Applications

Probability, statistical and mathematical computations

Generation of the Fibonacci sequence.

Generate and print the first n terms of the fibonacci series where $n \geq 1$

The first few terms are

0, 1, 1, 2, 3, 5, 8, 13 ...

Each term beyond the first two is derived from the sum of its two nearest ~~passed~~ predecessors

new term = preceding term + term before preceding term

Algorithm description

1. Prompt and read n, the number of fibonacci numbers to be generated.
2. Assign first two fibonacci numbers a and b
3. Initialize count of number generated.
4. While less than n Fibonacci numbers have been generated do
   (a) write out next two Fibonacci numbers;
   (b) generate next fibonacci numbu keeping a relevant;
   (c) generate next fibonacci number from most recent pair keeping b relevant for next computation
   (d) update count of numbu of fibonacci numbers generated;
5. If n even then write out last two fibonacci numbers else write out second last ~~fibonacci~~ fibonacci number

Pascal Implementation

```
Program Fibonacci(Input, output);
var a{fibonacci number variable};
    b{fibonacci number variable};
    i{number of fibonacci numbers generated};
    n{number of fibonacci numbers to be generated};
begin{generate each fibonacci number from the sum of its
      two predecessors}

    a:=0;
    b:=1;
    i:=2;
writeln('enter n the number of fibonacci numbers to be
                    generated');
readln(n);
  while i<n do
    begin
      writeln(a,b);
      a:=a+b;
      b:=a+b;
      i:=i+2;
    end;
  if i=n then writeln(a,b) else writeln(a).
end.
```

11/6/15

- To generate n fibonacci numbers essentially n steps are required. This algo functions correctly for all values of n ≥ 1
- Throughout the computation the variables a and b always contain the two most recently generated fibonacci numbers.

- Applications.

The fibonacci sequence has practical applications in botany, electrical network theory, sorting and searching.

Reversing the digits of an Integer

- Design an algorithm that accepts a positive integer and reverses the order of its digits
- Digit reversal is a technique that is sometimes used in computing to remove bias from a set of numbers.
- It is important in fast information-retrieval algorithms.

Algorithm description

1. Establish n, the positive integer to be reversed
2. Set the initial condition for the reversed integer (reverse).
3. While the integer being reversed is greater than zero do
   (a) use the remainder to extract the rightmost digit of the number being reversed
   (b) increase the previous reversed integer representation (reverse) by a factor of 10 and add it to the most recently extracted digit to give the current reverse value.

(c) use integer division by 10 to remove the rightmost digit from the number being reversed.

Pascal implementation

```
function reverse(n: integer): integer;
  var reverse : integer;
  begin{reverse the order of the digits of a positive integer}
   while n>0 do
     begin
       reverse := reverse *10 + n mod 10
         n := n div 10 ;
      end;
  end
```

Notes

1. The number of steps to reverse the digits is directly proportional to the number of digits in the integer

Applications

Hashing, and information retrieval, data base applications.

exp examples

→ design an algo that counts the number of digits in ~~an~~ an integer

→ design an algo to sum the ~~the~~ digits in an integer