Data Struct.

Data Structure :-

Organised Collection of data is known as Data structure.

```
Struct data
{
int Roll;
Char Name[30];
};
```

OR

The logical and mathematical representation of data is called Data structure.

There are two types of data structure

1) Linear data Structure — The linear data structure are the single level data structure in which the data item are arranged in a sequential (linear) form.

e.g:- Stack, Queue, linked list

Linear — Array
Stack
Queue
Linked list

ii) Non-linear data structure
The non-linear data structure are the single level structure in which data items are not arranged in a linear or sequence form.
e.g:- Tree, Graph

Non-linear DS
- Tree
- Graph

Stack

Stack is a linear data structure that follow the technique LIFO (Last in first out) means the element which is inserted at last comes out first. In the stack the insertion and deletion always take place from one end called Top. An insertion in a stack is called PUSH and deletion from stack is called POP operation.

When a stack is

implemented as an array it acquires the all capability of array and if implemented as a array linked list, all the characters are possessed by it.

Oper^{n} :— Push , Pop , peep
(Insert) (Delete) (Search दिखाना)

Algorithm for PUSH Operation

Step 1:— If (TOP == Size-1)
{
Print ("Stack Overflow");
Return ;
}

Step 2:— TOP = TOP + 1 ;
Step 3:— Stack [Top] = Value
Step 4:— Return

Algorithm for POP operation

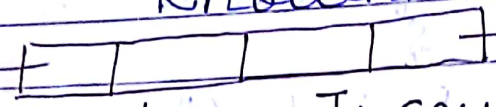
Step 1:— If (TOP == -1)
{
Print ("Stack Underflow");
Return ;
}

Step 2:— Val = Stack [Top]
Step 3:— Top = Top - 1
Step 4:— Return Val.

PAGE:
DATE: / /

Queue

The Queue is a linear data structure that follows the technique FIFO (First In First Out). In the Queue data structure the elements are inserted and deleted from two different ends. The position at which element is inserted in the Queue is known as Rear and the position from element is deleted known as front.



Algorithm for Insert Operation

Step 1:— If (Rear >= Size-1)
{
Print ("Queue overflow")
Return

Step 2:— if (front = Rear = -1)
{
Set (Rear = -1)
front = 0 ;
}

Step 3:— Rear = Rear + 1 ;
Step 4:— Q[Rear] = Val
Step 5:— Return

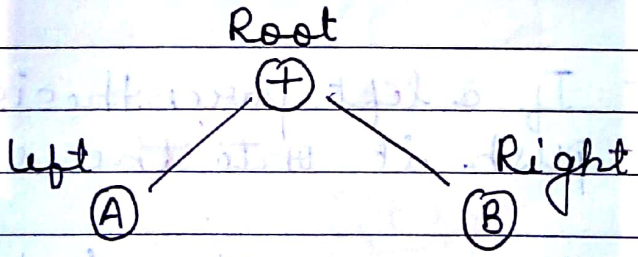
* Algorithm for Delete operation

Step 1:- If (front = Rear = 1)
{
print ("Queue Underflow")
Return ;
}

Step 2:- if (Front == Rear)
{
Val = Q [front]
Set
Front = -1 ;
Rear = -1 ;
Return Val ;
}

Step 3:- Val : Q [front]
Step 4:- front = front + 1
Step 5:- Return Val.

Infix to postfix conversion Using stack.



(i) Infix : left - Root - Right
A + B

(ii) Prefix : Root - left - Right
+ AB

(iii) Postfix : Left – Right – Root
A B +

Q Conversion of infix expression into postfix or prefix. using stack

→ Algorithm to convert infix to postfix or prefix using stack.

// Algorithm to convert infix to postfix.

Step 1:— PUSH "(" on to the stack and add ")" to the end of expression.
i.e., X

Step 2:— Scan the expression (X) from left to right and repeat step 3 to 6 for each element of X until the stack is empty.

Step 3:— If an operand is found, add it to the Y (expression).

Step 4:— If a left parenthesis is found, push it onto the stack.

Step 5:— If an operator is found, then

a) Repeatedly POP from stack and add to Y each operator (on the top of stack) which has the same

presidence as or higher presidence than operator.

b) Add Operator to stack.

Step 6:- If a right parenthesis is found then

a) Rapitidally POP from stack and add to Y each operator (on the top of stack) until a left parenthesis is found.

b) Remove the left parenthesis (do not add the left paranthesis to Y).

((A+B)-(C*(D/E)))

Input	Stack	output
(	(	
(	((	
A	((	A
+	((+	A
B	((+	AB
)	((+) POP	AB+
-	(-	AB+
(	(-(	AB+
c	(-(	AB+C
*	(-(*	AB+C
(	(-(*(	AB+C

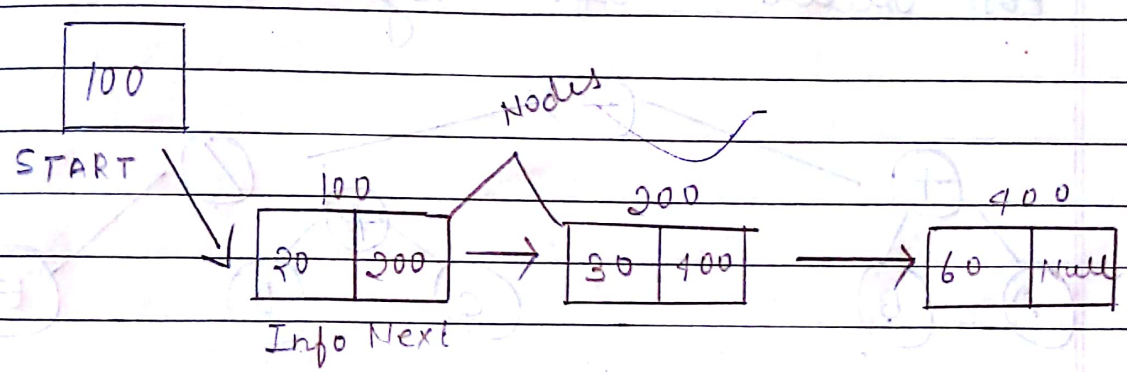
PAGE:
DATE: / /

Input	Stack	Output
D	(-(*(	AB+CD
/	(-(*(/	AB+CD
E	(-(*(/	AB+CDE
)	(-(*(/) , POP	AB+CDE/
)	(-(*) , POP	AB+CDE/*
)	(-)	AB+CDE/*-

linked list

linked list is a linear collection of data elements called nodes pointing to the next node by means of pointer.

In the linked list, the logical order in is represented by its pointing to the next element i.e., known as Node. Each node has two parts.



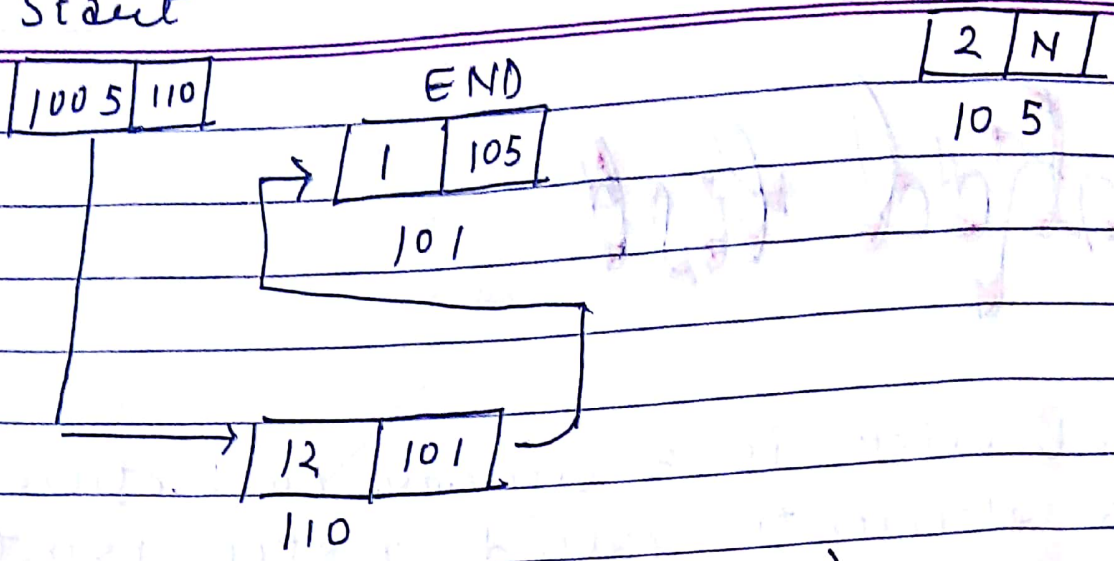
Single linked list.

Algorithm to insert a node at a start or beginning of linked list

i) Create a node
ii) ptr → Val = new value
iii) ptr → next = NULL
iv) ptr → next = start
v) start = ptr.

PAGE: 9
DATE: / /

Start

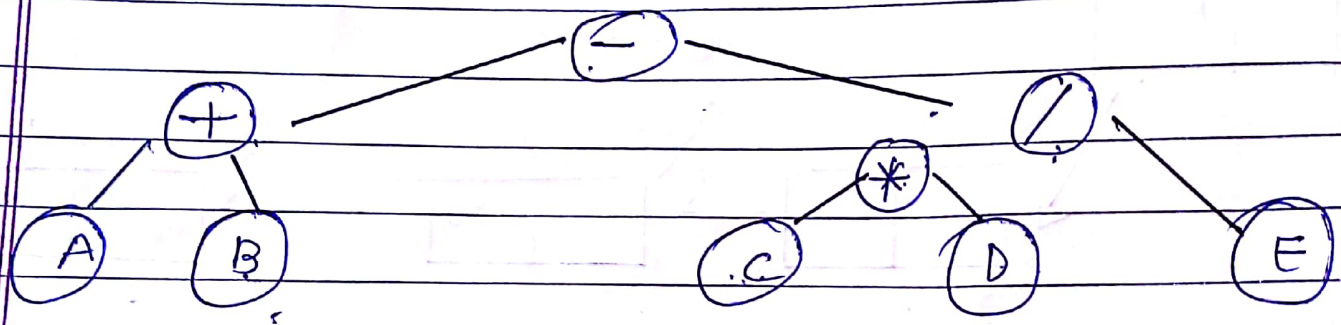


Tree (Binary tree)

Q) $5+2-(3*8/2)+5$

Q) $A+B-C*D/E$

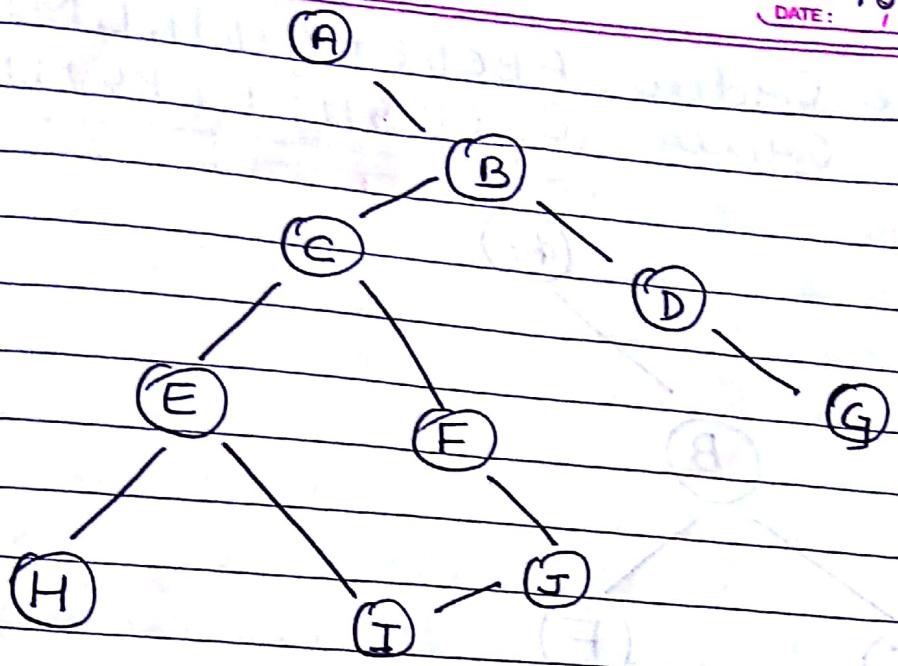
In Order	Left	Root	Right
Pre Order	Root	left	Right
Post Order	Left	Right	Root



In order : $A+B-C*D/E$

Pre order : $-+AB/*CDE$

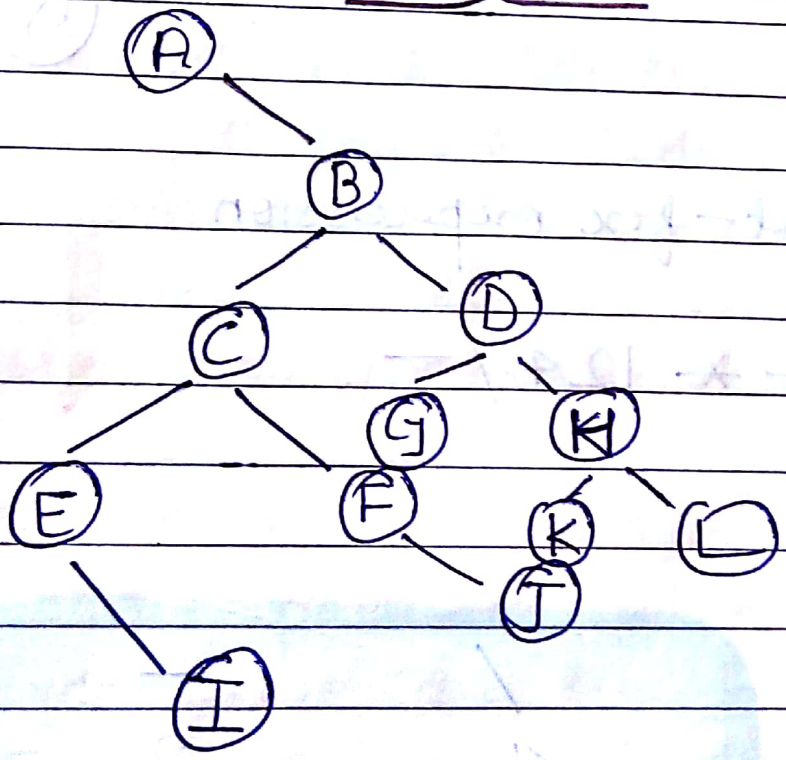
Post order : $AB+CD*E/-$



Pre Order A B C E H I F J D G
In Order A H E I C F B D G

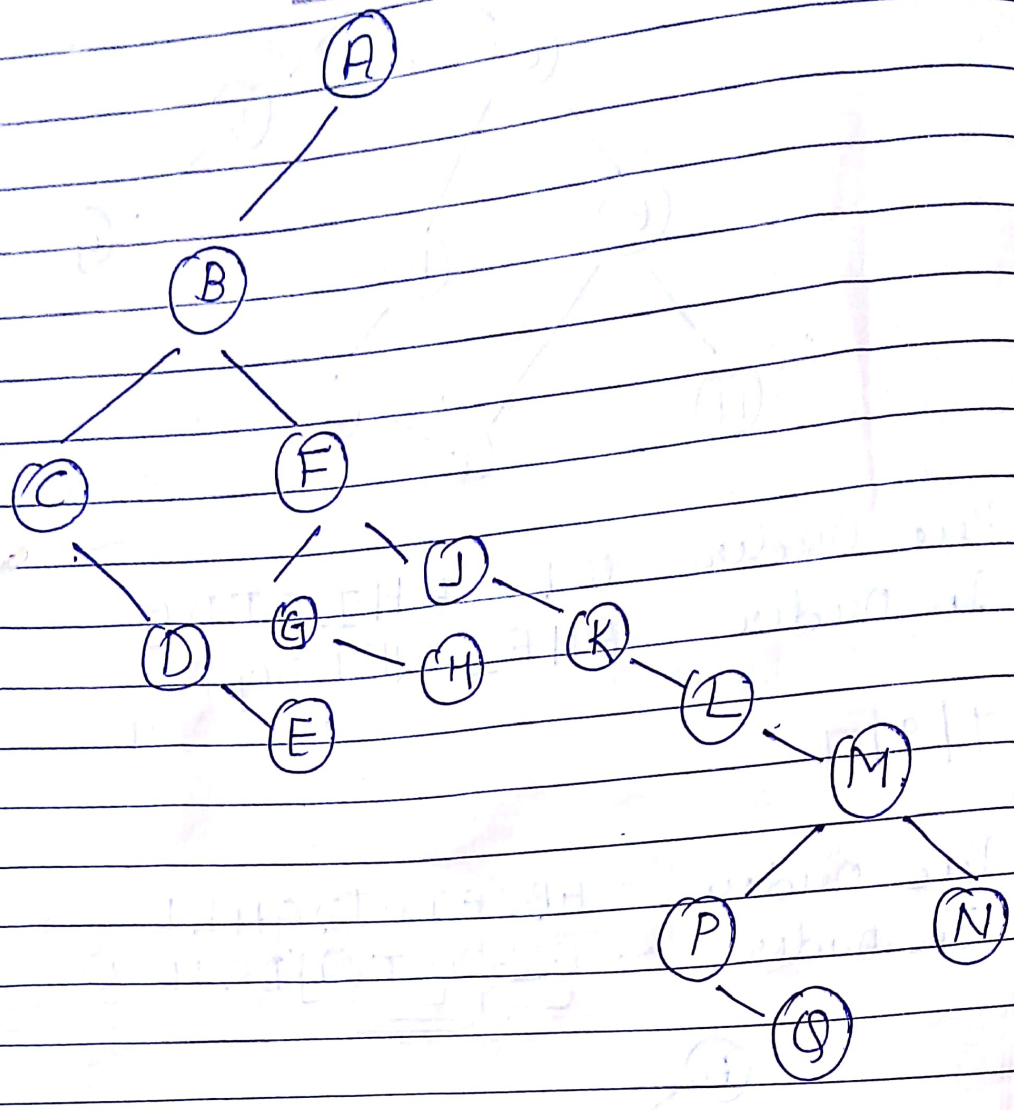
7/9/17

Pre Order – A B C E I F J D G H K L
In Order – E I C F J B G D K H L A



Pre Order A B C D E F G H J K L M P Q N

In Order C D E B G H F K L P Q M N J A



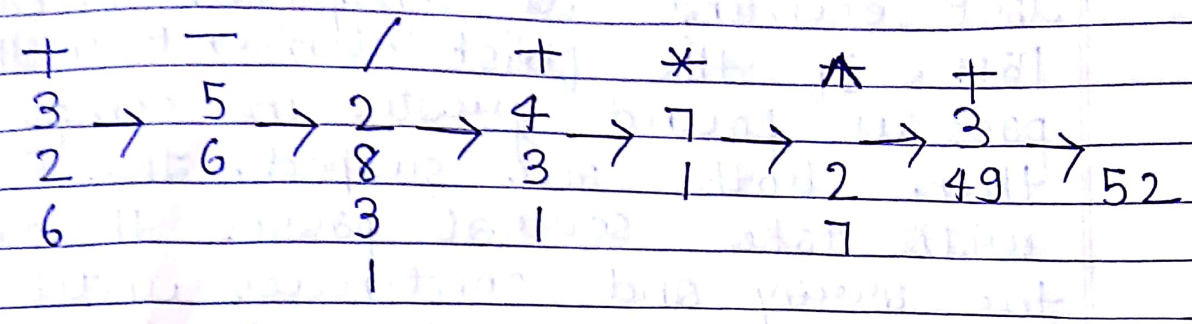
Evaluate post-fix expression

E = 5 6 2 + * 1 2 4 / −

Soln:-

+	*	/	−	
2	8	4	3	
6	5	12	40	37
5		40		

Q) 623+−382/+*2^3+



Sorting

The Sorting is the process of arranging data elements of the array in a particular order either in ascending or descending.

There are various types of Sorting technique available :-
Such as : Bubble Sort
Insertion Sort
Quick Sort
Selection Sort
Merge Sort.

1) **Bubble Sort :-** The bubble sort technique is useful for smaller size of array. In this technique successive pairs of elements are compared in successive order i.e., first element of the array is compared with the second element, then second element with third element and so on. This

process will be continued until the last element is compared in each pass. If the first element in the pass is found greater in comparison then both are swaped. The sorting will take several passes through the array and continues until the entire list of element is sorted.

In every pass, the first largest element is placed at the last position and second largest in second last position and so on.

Q)

0	10	5	5	5	5	
1	5	10	3	3	3	Pass I
2	3	3	10	2	2	
3	2	2	2	10	10	
4	12	12	12	12	12	

0	5	3	3	3	
1	3	5	2	2	
2	2	2	5	5	Pass II
3	10	10	10	10	
4	12	12	12	12	

0	3	2	2	
1	2	3	3	
2	5	5	5	Pass III
3	10	10	10	
4	12	12	12	

0	2	2
1	3	3
2	5	5
3	10	10
4	12	12

Pass IV

Ok Q.

5	4	4	4	4	4	4
4	5	5	5	5	5	5
6	6	6	6	6	6	6
7	7	7	7	1	1	1
1	1	1	1	7	2	2
2	2	2	2	2	7	7
9	9	9	9	9	9	9

Pass I

4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	1	1
1	1	1	1	6	2
2	2	2	2	2	6
7	7	7	7	7	7
9	9	9	9	9	9

Pass II

4	4	4	4	4	4
5	5	5	1	1	1
1	1	1	5	2	2
2	2	2	2	5	5
6	6	6	6	6	6
7	7	7	7	7	7
9	9	9	9	9	9

Pass III

4	4	1	1
1	1	4	2
2	2	2	4
5	5	5	5
6	6	6	6
7	7	7	7
9	9	9	9

Pass IV

Algorithm for bubble Souting

Suppose there is an array named AR containing N number of element, to sort the array in ascending order using bubble sort, the following steps to be followed.

STEP 1: Set I=0

STEP 2: Repeat step 3 to 7 while (I<N-1)

STEP 3: Set J=0

STEP 4: Repeat step 5 to 6 while (J<N-1)

STEP 5: If (AR [J]>AR [J+1])
{
Temp = AR [J]
AR [J] = AR [J+1]
AR [J+1] = Temp
}

STEP 6 : J = J+1

STEP 7 : I = I+1

STEP 8 : END

2) Insertion Sort :-

1) ~~5~~ ~~4~~ ~~6~~ ~~7~~ ~~1~~ ~~2~~ ~~9~~

5 4 6 7 1 2 9

4 5 6 7 1 2 9

4 5 6 7 1 2 9

1 4 5 6 7 2 9

1 2 4 5 6 7 9

1 2 4 5 6 7 9

3) Quick Sort :—

5 4 6 7 1 2 9

(5) 4 6 7 1 (2) 9

2 4 (6) 7 1 (5) 9

2 4 (5) 7 (1) 6 9

2 4 1 (7) (5) 6 9

2 4 1 (5) 7 6 9

2 4 1 (5) 7 6 9

sub list I sub list II

(2) 4 1 5 7 6 9

1 4 (2) 5 7 6 9

1 2 4 5 (6) 7 9

1 2 4 5 6 7 9

Searching

Searching is the process of determining whether the such element is present in the array or not.

There are two types of popular search techinque there are :—

(1) Linear searching :— In the linear searching method, the element to be searched is entered and the linear search starts scanning from the very first element sequentially and compaires each element of the array with the search element. If match is found, the control immediatly comes out from a loop with a message "Element is found". If search element is not found the control return at last after comparing each element of the array with a not found message.

The linear search is applicable on sorted or unsorted less size of array.

DATE: / /

Algorithm for linear search :-

Step 1 :- Declare an array A[] of size found = 0, i = 0.

Step 2 :- Insert array elements

Step 3 :- Input the value val to search.

Step 4 :- Repeat step 5 until i < N

Step 5 :- if (val == A[i])
found = 1
Return.

Step 6 :- If found == 1
Print ("Value in list")
else
Print ("Value not in list")

Step 7 :- Stop

Step 8 :- End.

(2) Binary Search :— The binary searching method is very popular in searching of item in min^m possible comparissions. The binary search requires the array must be in sorted order either in ascending or decending.

To search an item in a sorted order, the item is compared with middle element of the segment if the item is more than the middle element, the letter part of the segment becomes new segment to be searched. If the item is less than the middle element, the former part of the segment becomes new segment to be searched. The same process is repeated for the new segment until either the item is found or the segment is reduced to the single element and still the item is not found.

Algorithm for Binary Search:—

Step 1:— Declare an array A[] of size N, min=0 max = N-1, found=0.
Step 2:— Insert array element.
Step 3:— Input the value val to search.
Step 4:— Repeat step 5,6,7 until min<max.

Step 5 :- mid = (min + max) / 2

Step 6 :- if (A[mid] == Val)
{
found = 1
Return ;
}

Step 7 :- if (Val < A[mid])
max = mid - 1
else
min = mid + 1

Step 8 :- if (found == 1)
print ("Value in list")
else
print ("Value not in list")

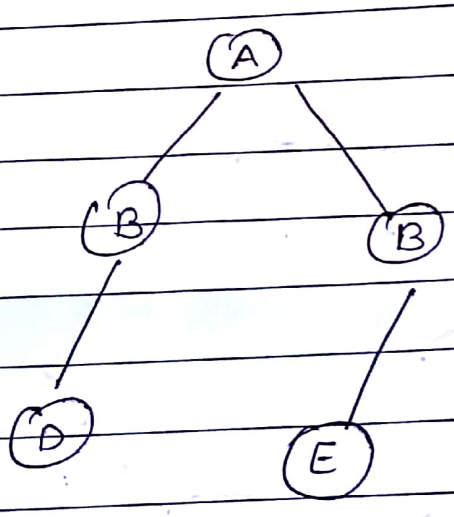
Step 9 :- STOP

Step 10 :- END.

Graph

A graph G consist of a set V of vertices (nodes) and a set E of edges (arcs). We write G = (V, E), V is finite and non empty set of vertices. E is a set of pairs of vertices, their pair are called edges.

Degree :- NO. of edges



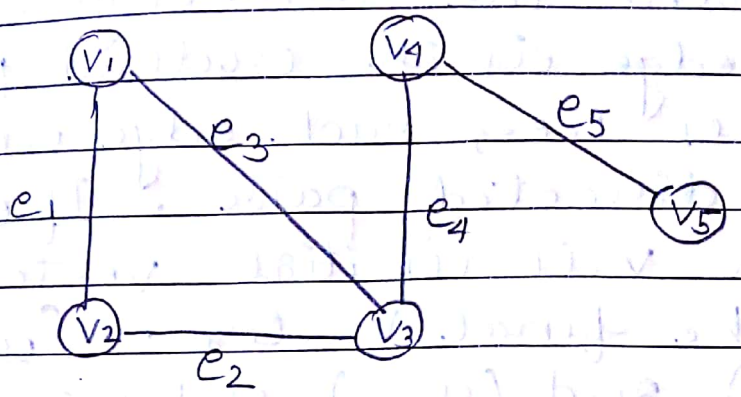
Therefore

V (G), read as V of G, is a set of vertices.

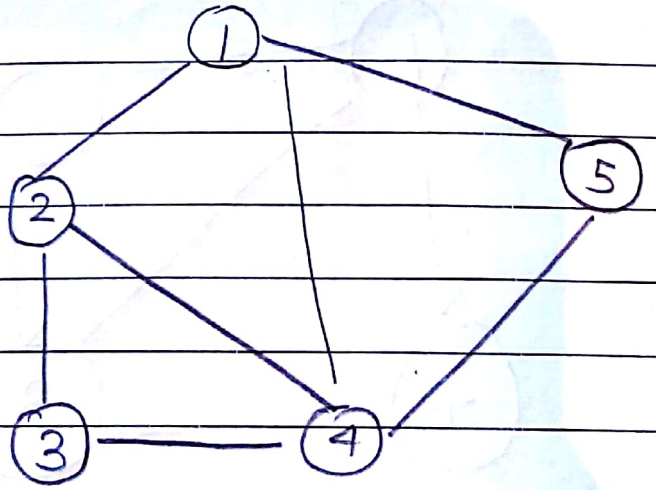
E (G), read as E of G, is a set of edges.

and
An edge $e = (v, w)$, is a pair of vertices v and w and is said to be incident with v and w.

$$V(G) = \{v_1, v_2, v_3, v_4, v_5\}$$
and $$E(G) = \{e_1, e_2, e_3, e_4, e_5\}$$



i.e., there are five vertices and five edges in the graph.

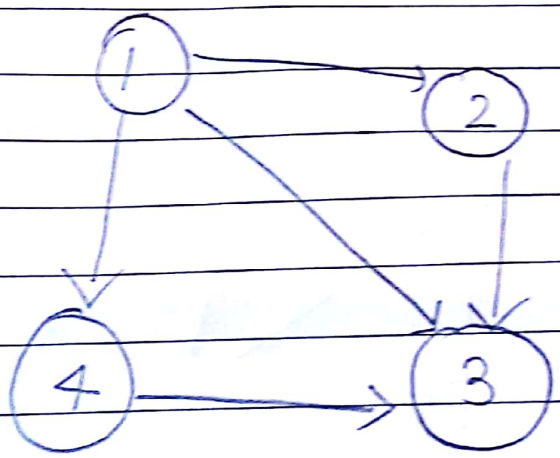


We have numbered the nodes as 1, 2, 4 and 5. Therefore

$$V(G) = (1, 2, 3, 4, 5)$$
and $$E(G) = \{(1,2), (2,4), (2,3), (1,4), (1,5), (4,5), (3,4)\}$$

1) Undirected graph – out degree
2) directed graph – in degree

In an Undirected graph, pair of vertices representing any edge is unordered. Thus (v, w) and (w, v) represent the same edge. In the each edge is an ordered pair of vertices i.e., each edge is represented by a directed pair. If e = (v, w), then v is initial vertex and w is the final vertex. Subsequently (v, w) and (w, v) represent two different edges. A directed graph may be pictorically



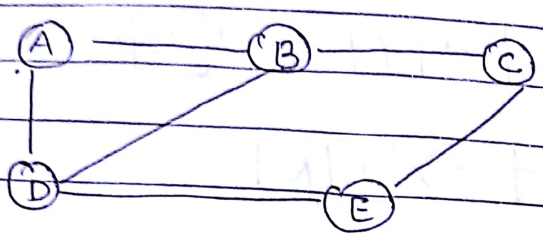
Directed Graph

PAGE:
DATE: / /

Representation of Graph (using array)

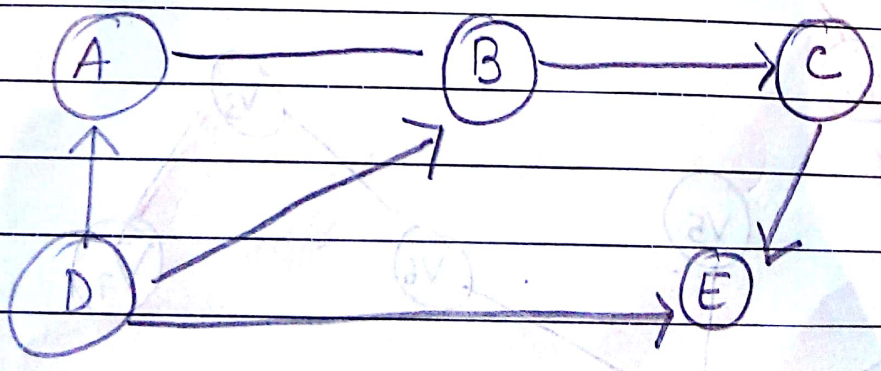
(Adjacency)

1) Adjacency Matrix :- It is used to represent which nodes are adjacent to one another.

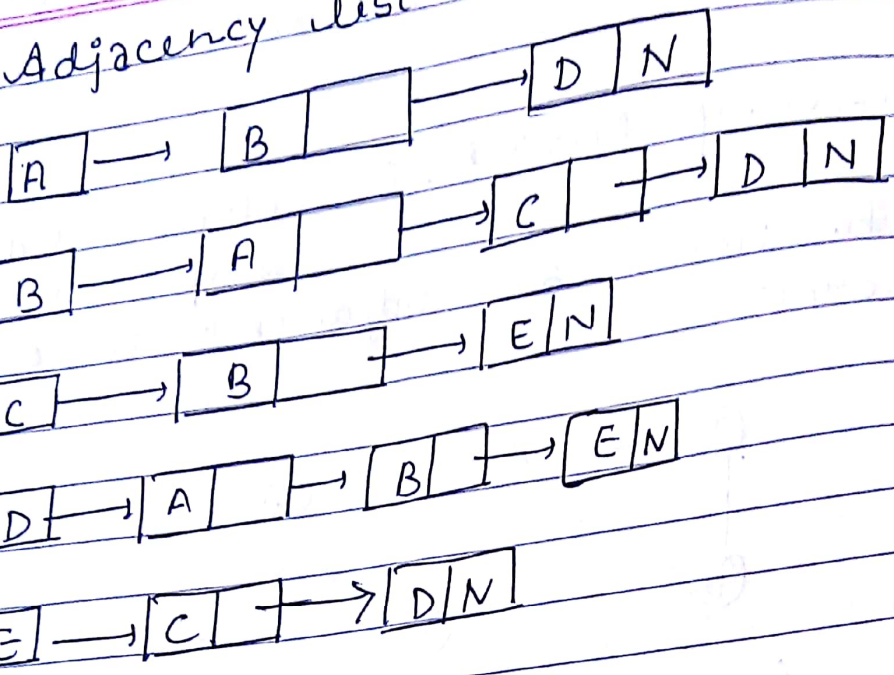


	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

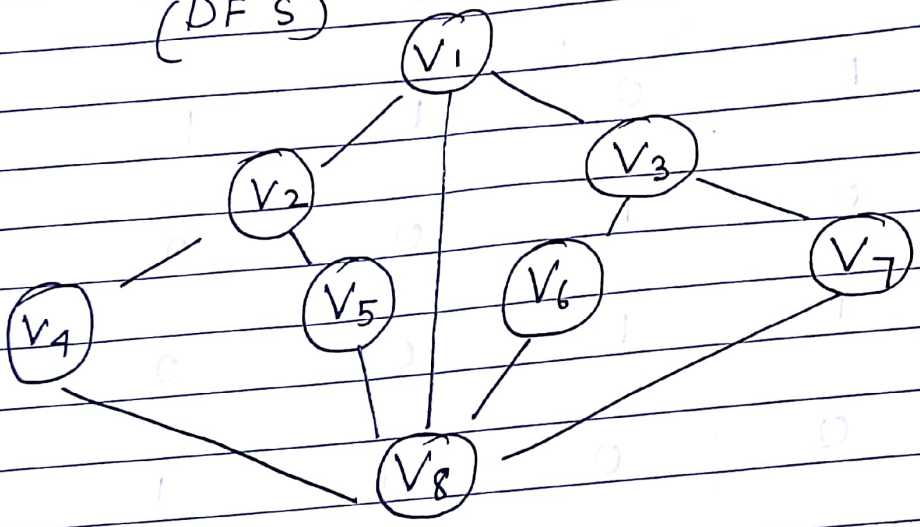
link Representation of a graph (adjecency list representation):-

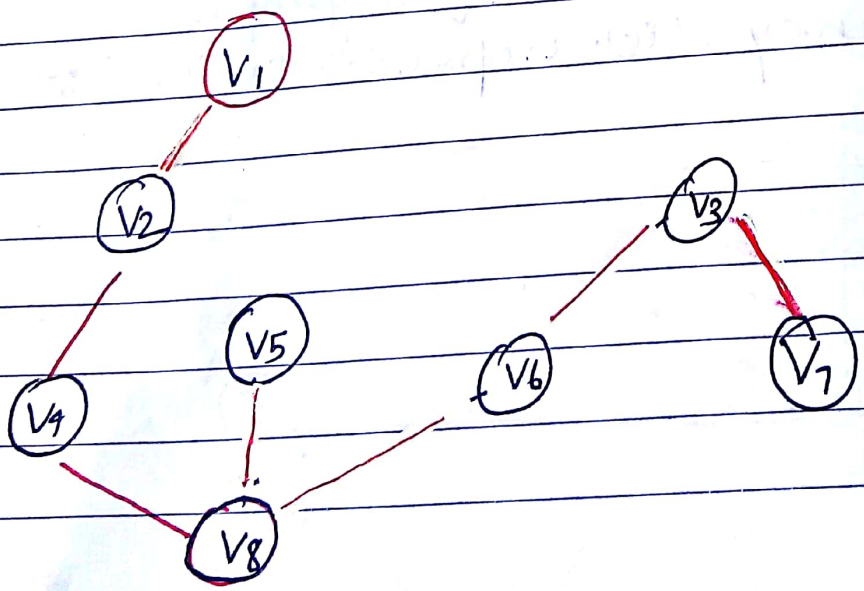


Adjacency list



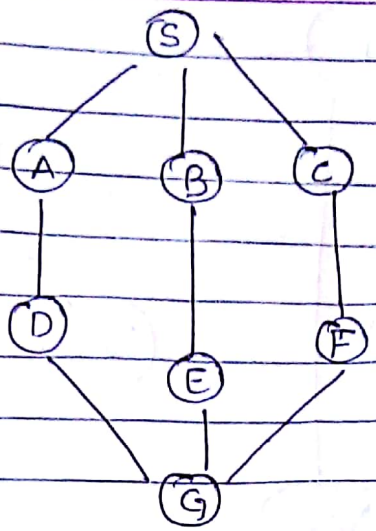
* Depth first Search :—
(DFS)

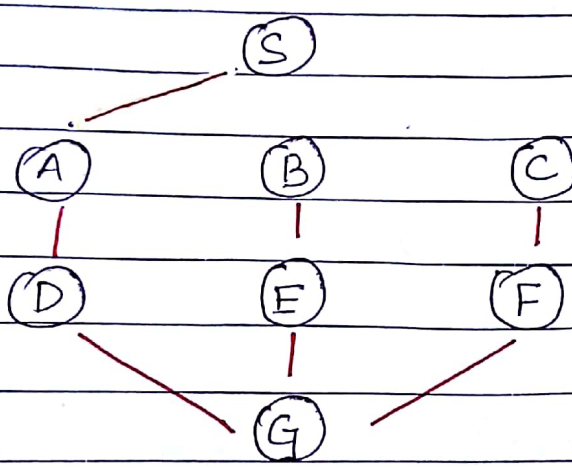




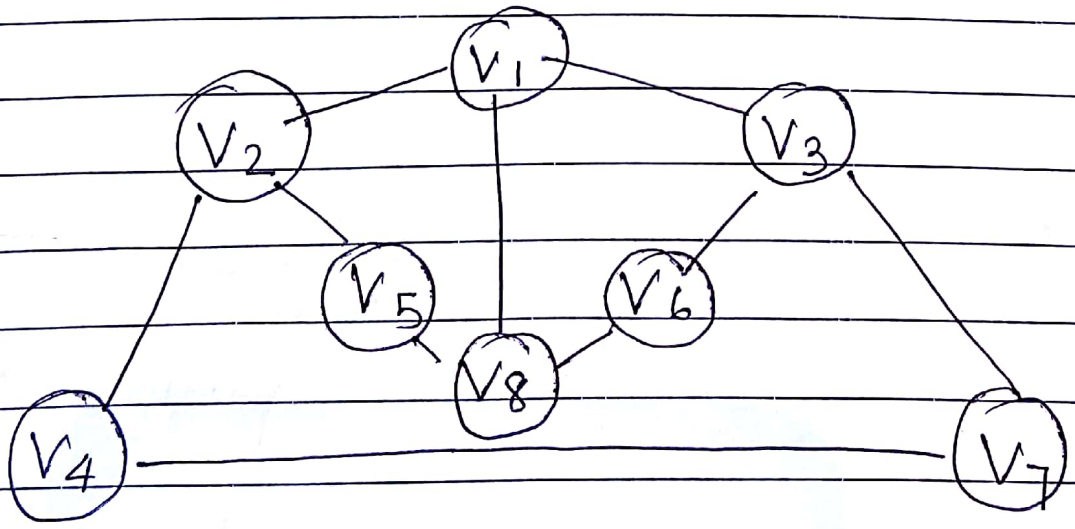
PAGE:
DATE: / /

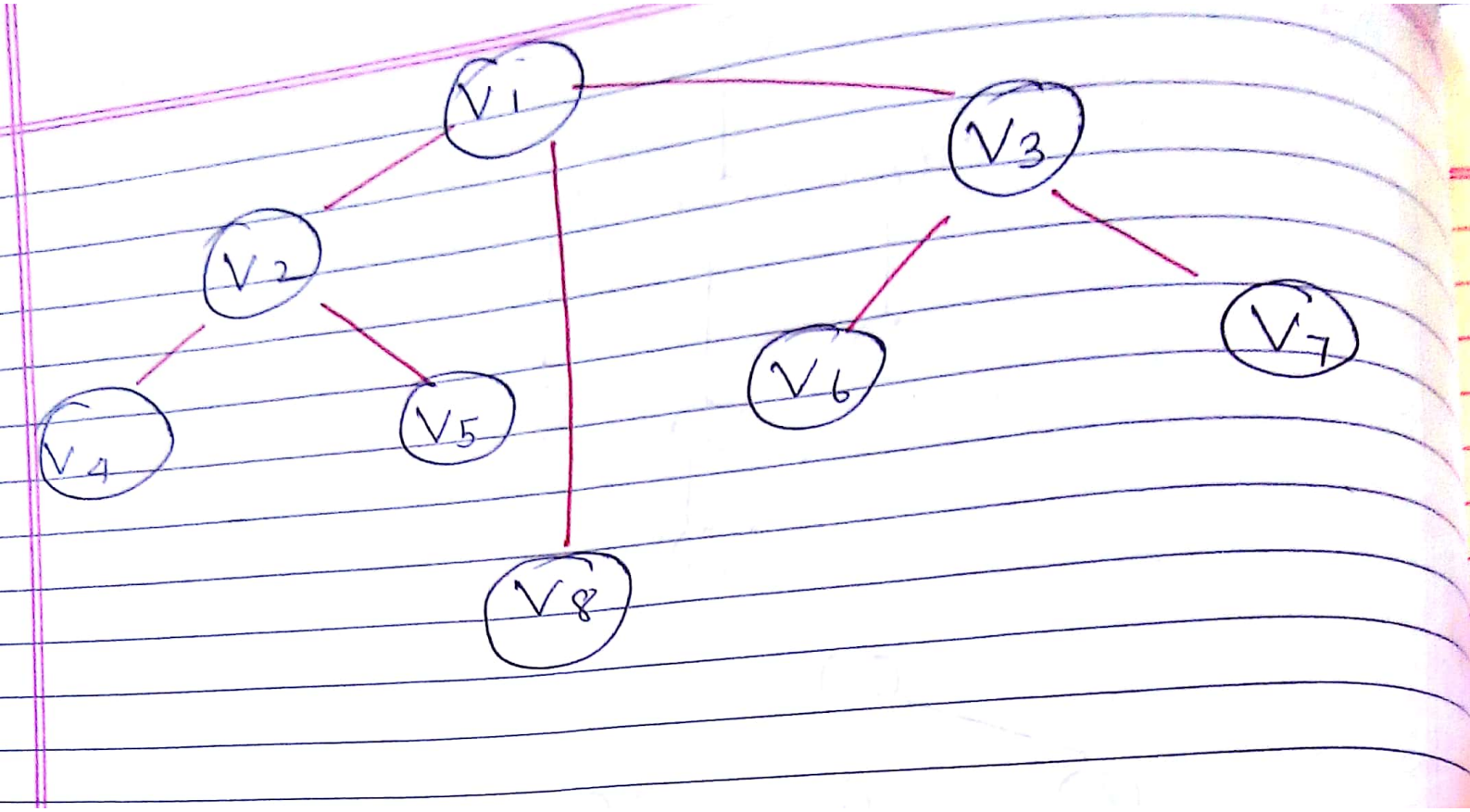
Q)





BFS → Breadth first Search





13/12/17

1. Write an algorithm for insert and delete operation in a circular queue.

Step 1:- If (front == (Rear+1) % MAXSIZE)
Print ("Queue Overflow")
and exit.

Step 2:- Else
Take the value
If (front == -1)
set (front = 0)
(Rear = -1);

Step 3:- Rear = ((Rear + 1) % MAXSIZE)

Step 4:- Queue [Rear] = value

Step 5:- End

14/12/17

→ Algo to insert at start / begining of singly linked list

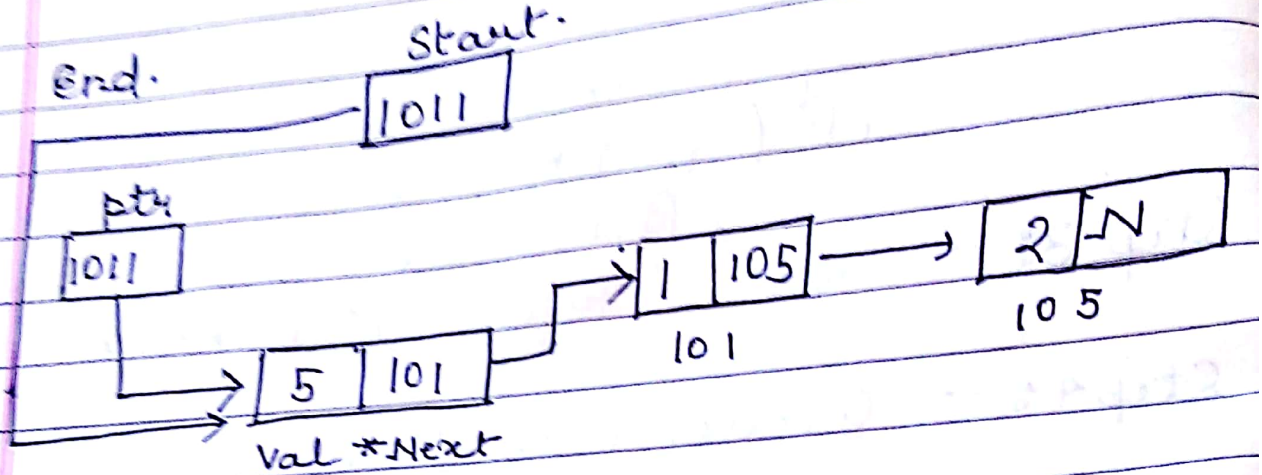
Begin
Step 1:- if (ptr == NULL)
{
print ("Overflow");
Return;
}

else
{
ptr=(NODE *) malloc (size of (NODE))

step 2:- ptr → Val = item

step 3:- ptr → next = start

step 4:- Start = ptr



* Deletion of a node from start of a linked list :-

Begin

step 1:- Start == NULL
{
print ("linked list empty")
Return ;
}

p 2:- ptr = Start

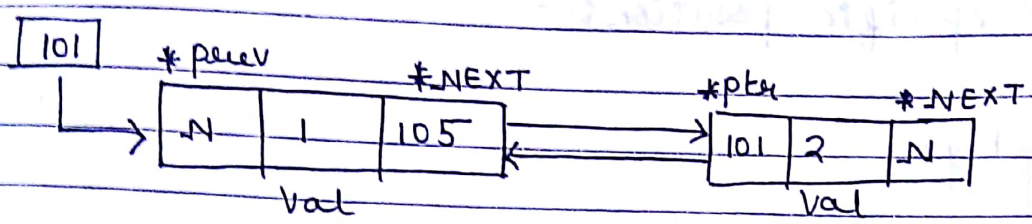
p3:- Start = start → Next

p4:- free (ptr)

end

PAGE:
DATE: / /

* Structure of a doubly linked list :-

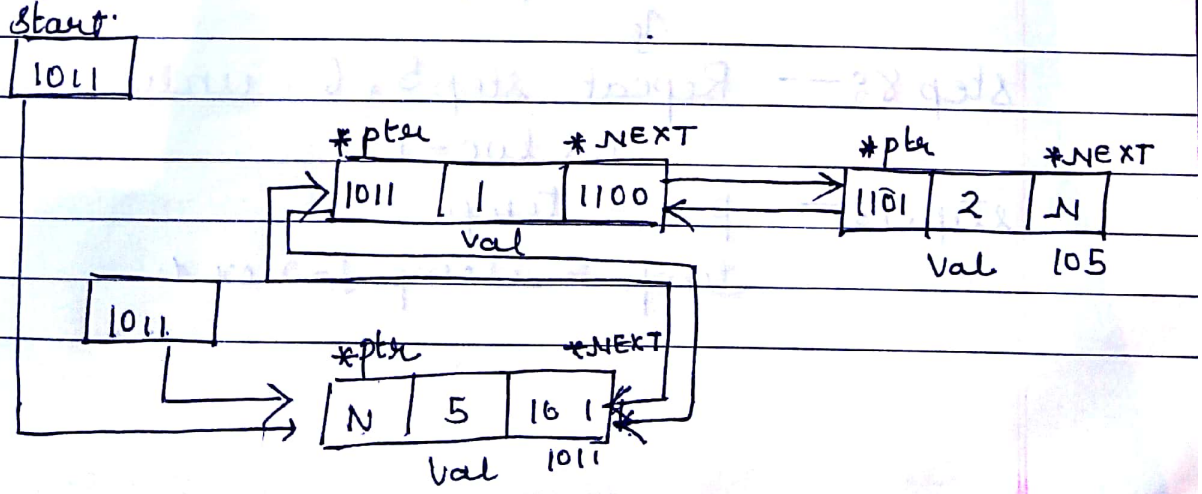


algo to insert of start begining of doubly linked list.

Begin

step 1:- if (ptr == NULL)
{
print ("Overflow")
Return;
}
else
{
ptr = (NODE *) malloc (size of (NODE))

step 2:- ptr → val = item
step 3:- ptr → next = NULL
step 4:- ptr → prev = NULL
step 5:- start → prev = ptr
step 6:- ptr → next = start
step 7:- start = ptr.



15/12/17 (friday)

Write an algo for insert of a node at specific position:-

Begin

```
step 1:- if (ptr == NULL)
         {
         print ("Overflow")
         Return;
         }
         else
         {
         ptr=(NODE *) malloc (size of node)
step 2:- ptr → Val → item
step 3:- ptr → Next → Start
step 4:- if (start == Null)
         {
         start = ptr;
         }
step 5:- Insert location loc, i=0
step 6:- temp = start, ptr1= temp
step 7:- if (loc-1==0)
         {
         ptr → next = start
         stack = ptr
         }
step 8:- Repeat step 5, 6. until
         I < loc-1
step 9:- ptr = temp
         temp = temp 1=next.
```

PAGE:
DATE: / /

Step 10:— temp = temp → next, i++
ptr 1 ⇒ next = ptr
ptr → next = temp

Step 11 :— end.