1. Write an algorithm for PUSH and POP operations using arrays.

Operation	Description	Restriction
PUSH	This adds(or pushes) a new data item on to the stack	The number of data items on the stack should not exceed MAXSIZE.
POP	This deletes the data item from the top of the stack	The number of data item on the stack should not go beyond zero
TOP	It returns the *value* of the item at the top of the stack	_
Stack_empty	It returns *true* if the stack is empty. Otherwise it returns false	_
Stack_full	It returns *true* if the stack is Full. Otherwise it returns false	_

PUSH(STACK, TOP, MAXSTK, ITEM)

This procedure pushes an ITEM onto a stack.

1. [Stack already filled?]

   If TOP = MAXSTK, then: Print: OVERFLOW, and Return.

2. Set TOP: = TOP + 1. [Increases TOP by 1.]

3. Set STACK [TOP] := ITEM. [Inserts ITEM in new TOP position.]

4. Return.

POP (STACK, TOP, ITEM)

This procedure deletes the top element of STACK and assigns it to the variable ITEM.

1. [Stack has an item to be removed?]

   If TOP = 0, then: Print: UNDERFLOW, and Return.

2. Set ITEM := STACK[TOP]. [Assigns TOP element to ITEM.]

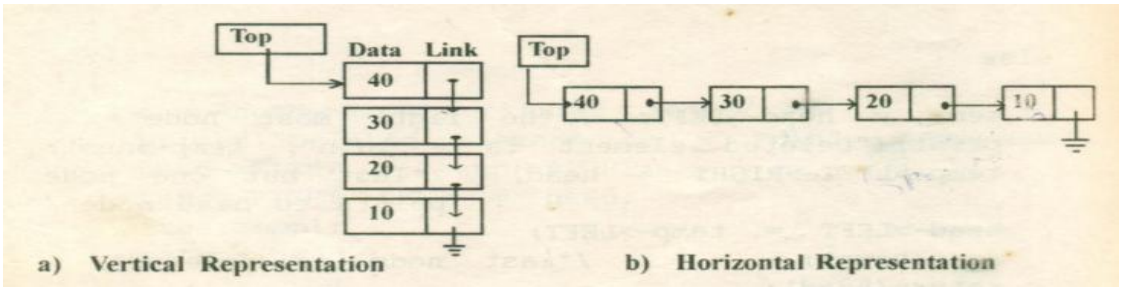
3. Set TOP := TOP - 1. [Decreases TOP by 1.]

4. Return.

2. Write an algorithm to push an item onto a stack using linked list.

Linked Stacks

A stack is a data structure in which all insertions and deletions are performed at one end called **top**. Insertion operation is called **pushing** and deletion operation is called **poping**.



PUSH operations performed on a linked stack

```
PUSH (NODE *top)
{
    NODE *newnode;
    int item;
    newnode = allocatenode();
newnode->num=item;
newnode->ptr = top;
    top = newnode;
    return(top);
}
```

Algorithm

STEP 1: CREATE A NEW NODE SET NEWNODE := new NODE

STEP 2: INPUT ITEM

STEP 3: SET NEWNODE->DATA:= ITEM

STEP 4: IF(TOP=NULL) THEN

a)SET TOP := NEWNODE

b)NEWNODE ->LINK := NULL

ELSE

a)SET NEWNODE -> LINK :=TOP

b)SET TOP := NEWNODE

[END OF IF STRUCTURE]

STEP 5: EXIT

3. Write an algorithm to delete an item of a stack using linked list.

POP operation performed on a Linked Stacks

```
POP (NODE *top)
{
    NODE *temp;
    if(top == NULL)
    {
            printf("stack is empty\n");
            return;
    }
    else
    {
            temp=top;
             top = top->ptr;
            printf("Deleted element is =%d\n", temp->num);
            free(temp);
    }
}
```

Algorithm

STEP 1: IF(TOP = NULL)THEN

WRITE: “STACK IS EMPTY”

ELSE

a)SET TEMP := TOP

b)SET TOP := TOP ->LINK

c)WRITE: TEMP ->DATA

d)DELETE (TEMP)

[END OF IF STRUCTURE]

STEP 2: EXIT

5.Write an algorithm to evaluate postfix expression

Evaluation of a Postfix Expression

Algorithm

I. Store the numeric values corresponding to each operand, in an array.

2. Scan the given postfix expression from LEFT to RIGHT.

a) If the scanned symbol is an operand (variable name), then push its value onto the stack.

b) If the scanned-symbol is an operator, then pop out two values from the stack and assign them respectively to operand2 and operand1

3.Then perform the required arithmetic operation.

case operator of

* : result = operand 1 * operand2

/ : result = operand 1 / operand2

+ : result = operand I + operand2

- : result = operand I - operand2.

end case

4. Push the result onto the stack.

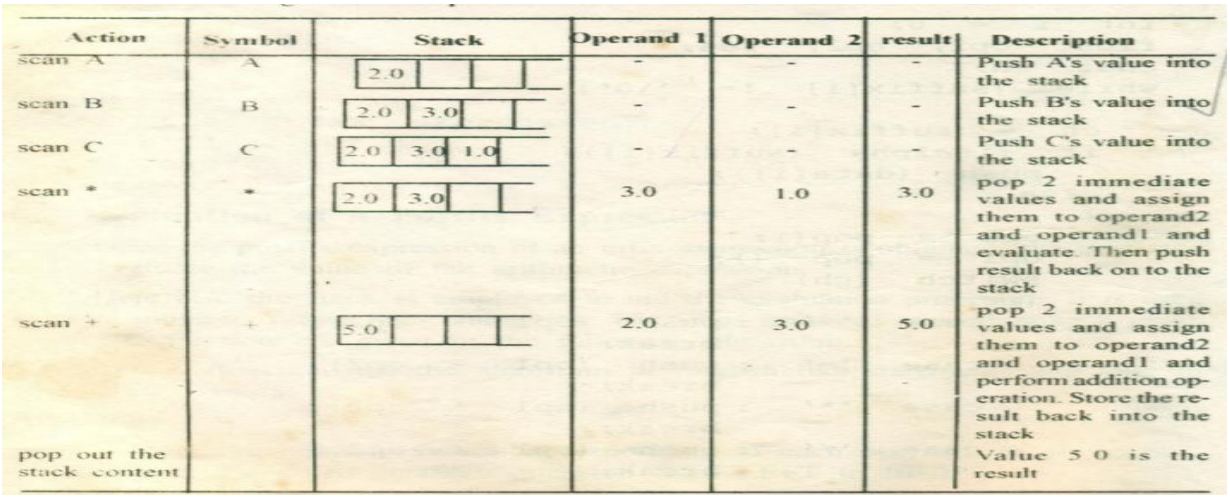
5. Repeat steps 1 to 4 until all the symbols in the postfix expression are over

Example

INFIX EXPRESSION=A+B*C

POSTFIX EXPRESSION=ABC*+

A=2.0 B=3.0 C=1.0



Action	Symbol	Stack	Operand 1	Operand 2	result	Description
scan A	A	2.0	-	-	-	Push A's value into the stack
scan B	B	2.0 3.0	-	-	-	Push B's value into the stack
scan C	C	2.0 3.0 1.0	-	-	-	Push C's value into the stack
scan *	*	2.0 3.0	3.0	1.0	3.0	pop 2 immediate values and assign them to operand2 and operand1 and evaluate. Then push result back on to the stack
scan +	+	5.0	2.0	3.0	5.0	pop 2 immediate values and assign them to operand2 and operand1 and perform addition operation. Store the result back into the stack
pop out the stack content						Value 5.0 is the result

6.Write an algorithm to convert infix to postfix expression.

Algorithm: INFIX TO POSTFIX

1. Scan the infix expression from LEFT to RLGHT

2. a) If the scanned symbol is left parenthesis, push it onto the stack.

b) If the scanned symbol is an operand, then place directly in the postfix expression.

c) If the scanned symbol is a right parentheses, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parentheses.

d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than or equal to the precedence of the scanned operator.

Otherwise, push the scanned operator on to the stack.

INFIX:A+B*C

POSTFIX:ABC*+

Action	Symbol	Stack	Postfix expn.	Description
scan A	A	empty	A	Place it on the postfix
scan +	+	+	A	Push + onto the stack
scan B	B	+	AB	Place it in the postfix
scan *	*	+*	AB	* has *precedence* over + so push onto the top of the stack
scan C	C	+*	ABC	Straight away place C in the postfix expression
All symbols are over	Nil	Nil	ABC*+	Popout all operators from the stack and place them in the postfix

12. Write an algorithm to insert an element into a queue using arrays.

Simple Queue

QUEUEINSERT (QUEUE, MAXSIZE, FRONT, REAR,ITEM)

This procedure inserts an ITEM into a queue

1. [Queue already filled?]

If (REAR==MAXSIZE) then

Write: OVERFLOW and Return

[End of If]

2. SET REAR: =REAR+1

3. QUEUE [REAR]:=ITEM

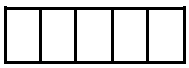
4. If (FRONT==0) then

SET FRONT: =1

[End of If]

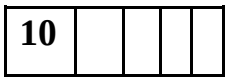
5.RETURN

Element Insert to the Queue



FRONT = 0

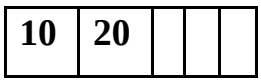
REAR = 0



FRONT =1 REAR=REAR+1

REAR = 1 ITEM=10 FRONT=1

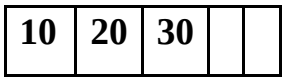
QUEUE[REAR]=ITEM



FRONT = 1 REAR=REAR+1

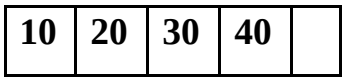
REAR = 2 ITEM=20

QUEUE[REAR]=ITEM



FRONT = 1 REAR=REAR+1

REAR = 3 ITEM=30 QUEUE[REAR]=ITEM



FRONT = 1 REAR=REAR+1

REAR = 4 ITEM=40

QUEUE[REAR]=ITEM

13. Write an algorithm to delete an element from a queue using arrays.

QUEUEDELETE (QUEUE,MAXSIZE,FRONT,REAR,ITEM)

This procedure delete an ITEM from a queue

1. [Check Queue is empty ?]

If (FRONT==0) then

Write: UNDERFLOW and Return

[End of If]

2. SET ITEM:=QUEUE[FRONT]

3. If(FRONT=REAR) then

a. SET REAR: =0

b.SET FRONT:=0

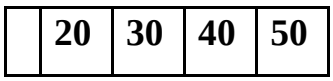
Else

SET FRONT:=FRONT+1

[End of If]

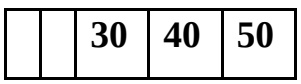
4.RETURN

Element Delete to the Queue



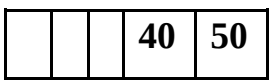
FRONT = 2
REAR = 5

ITEM=QUEUE[FRONT]
ITEM=10
FRONT=FRONT+1



FRONT = 3
REAR = 5

ITEM=QUEUE[FRONT]
ITEM=20
FRONT=FRONT+1



FRONT = 4
REAR = 5

ITEM=QUEUE[FRONT]
ITEM=30
FRONT=FRONT+1

				50

FRONT = 5 ITEM=QUEUE[FRONT]

REAR = 5 ITEM=40

FRONT=FRONT+1

FRONT = 0 ITEM=QUEUE[FRONT]

REAR = 0 ITEM=50

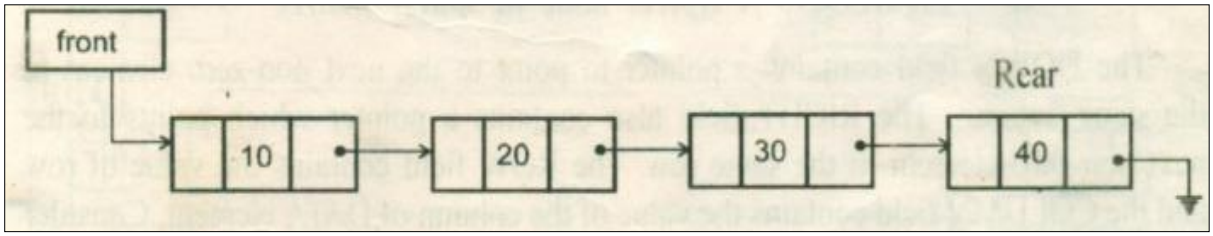
FRONT=0 REAR=0

14.Write an algorithm to insert an element into a queue using linked list.

Linked Queues

A queue is a data structure in which all insertions are performed at one end called the **rear** and all deletions are performed at the other end called the **front.**

In a linked queue, the insertion of a new node is done at the end of the linked list. And, the deletion operation is performed from the beginning of the linked list.



In this figure, **front** is an external pointer pointing to the very first node in the linked queue.

Linked Queues – Insert

STEP 1: CREATE A NEW NODE SET NEWNODE := new NODE

STEP 2: INPUT ITEM

STEP 3: SET NEWNODE -> DATA := ITEM

STEP 4: SET NEWNODE -> LINK = NULL

STEP 5: IF(FRONT = NULL)

a)SET FRONT := NEWNODE

b)SET REAR := NEWNODE

ELSE

a)SET REAR -> LINK := NEWNODE

b)SET REAR := NEWNODE

[END OF IF STRUCTURE]

STEP 6: EXIT

15. Write an algorithm to delete an element from a queue using linked list.

Linked Queues – Delete

STEP 1: IF(FRONT = NULL)THEN

WRITE :"QUEUE IS EMPTY"

ELSE

a)SET ITEM := FRONT -> DATA

b)SET FRONT := FRONT -> LINK

c).WRITE: "DELETE THE ITEM" ITEM

[END OF IF STRUCTURE]

STEP 2:EXIT

16.Write an algorithm to insert an element into a circular queue using arrays.

Algorithm

Insert

CIRCULAR_QUEUE_INSERATION(CQUEUE,MAXSIZE,FRONT,REAR,ITEM)

1. IF(FRONT:=(REAR+1)%MAXSIZE) THEN

WRITE: "Circular queue is full"

RETURN

[END OF IF]

2. SET REAR:=(REAR+1)%MAXSIZE;

3. SET CQUEUE[REAR]:=ITEM

4.IF(FRONT=-1) THEN

SET FRONT:=0;

[END OF IF]

5.RETURN

17. Write an algorithm to delete an element from a Circular queue using arrays.

Delete

CIRCULAR_QUEUE_DELETION(CQUEUE,MAXSIZE,FRONT,REAR,ITEM)

1. IF(FRONT=-1) THEN

WRITE "Element is empty"

RETURN

[END OF IF]

2. SET ITEM:=CQUEUE[FRONT]

3. IF(FRONT=REAR)THEN

a. SET FRONT:=-1;

b. SET REAR:=-1;

ELSE

SET FRONT:=(FRONT+1)%MAXSIZE

[END OF IF]

4.RETURN

18.Write a note on i) Priority Queue. Ii) Deque.

Priority Queues

A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules:

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with the same priority are processed according to the order in which they were added to the queue .

A prototype of a priority queue is a timesharing system: programs of high priority are processed first, and programs with the same priority form a standard queue.

There are various ways of maintaining a priority queue in memory. We discuss two of them here: one uses a one-way list, and the other uses multiple queues. The ease or difficulty in adding elements to or deleting them from a priority queue clearly depends on the representation that one chooses.

One-Way List Representation of a Priority Queue

One way to maintain a priority queue in memory is by means of a one-way list, as follows:

(a) Each node in the list will contain three items of information: an information field INFO, a priority number PRN and a link number LINK.

(b) A node X precedes a node Y in the list (1) when X has higher priority than Y or (2) when both have the same priority but X was added to the list before Y. This means that the order in the one-way list corresponds to the order of the priority queue.

Priority number will operate in the usual way: the lower the priority number, the higher the priority.

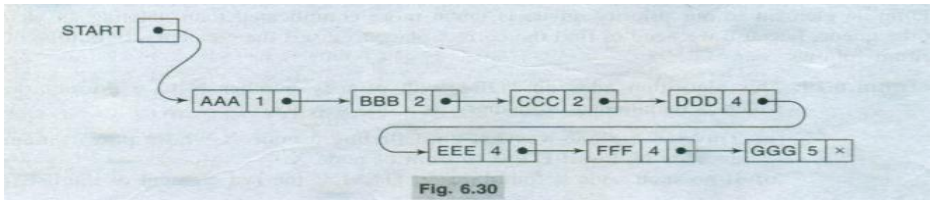


Fig. 6.30

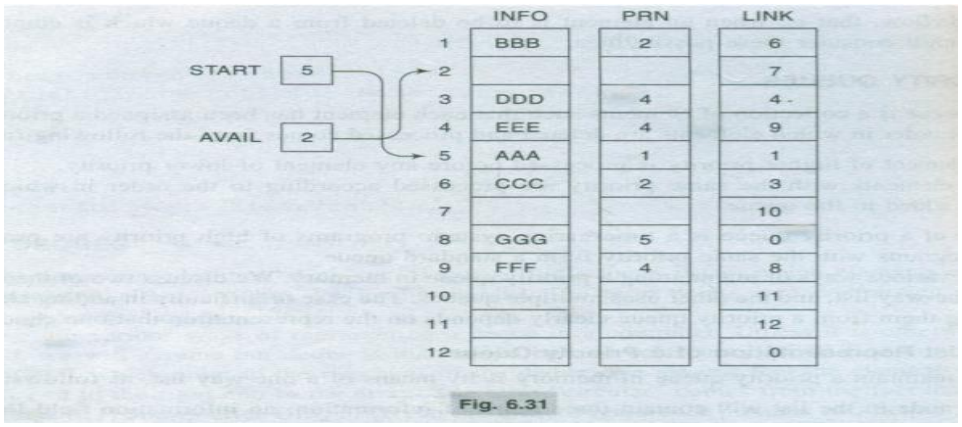


Fig. 6.31

Array Representation of a Priority Queue

Another way to maintain a priority queue in memory is to use a separate queue for each level of priority (or for each priority number). Each such queue will appear in its own circular array and must have its own pair of pointers, FRONT and REAR. In fact, if each queue is allocated the same amount of space, a two-dimensional array QUEUE can be used instead of the linear arrays. Figure 6.30 indicates this representation for the priority queue in Fig. 6.33. Observe that FRONT[K] and REAR[K] contain, respectively, the front and rear elements of row K of QUEUE, the row that maintain the queue of elements with priority number K.

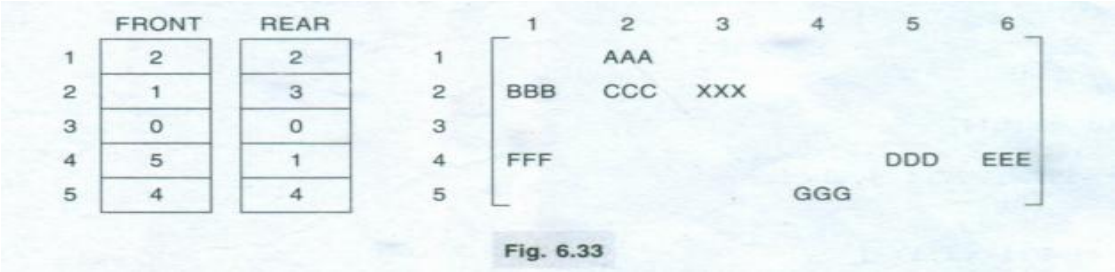


Fig. 6.33

(Double ended Queue)

A deque (pronounced either "deck" or "dequeue") is a linear list in which elements can be added or removed at either end but not in the middle. The term deque is a contraction of the name double-ended queue.

There are various ways of representing a deque in a computer. Unless it is otherwise stated or implied, we will assume our deque is maintained by a circular array DEQUE with pointers LEFT and RIGHT, which point to the two ends of the deque.

There are two variations of a deque-namely, an input-restricted deque and an output-restricted deque-which are intermediate between a deque and a queue. Specifically, an input-restricted deque is a deque which allows insertions at only one end of the list but allows deletions at both ends of the list; and an output-restricted deque is a deque which allows deletions at only one end of the list but allows insertions at both ends of the list.

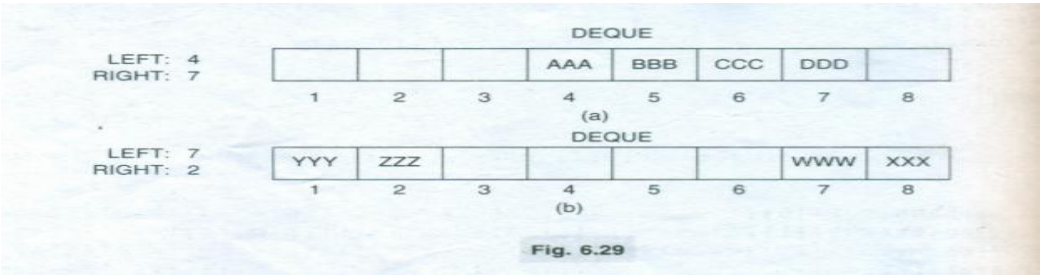
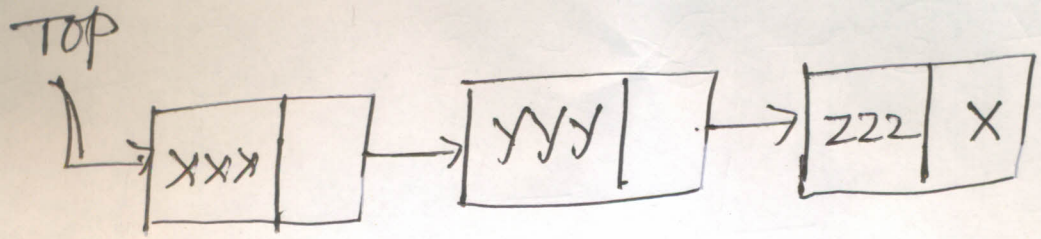


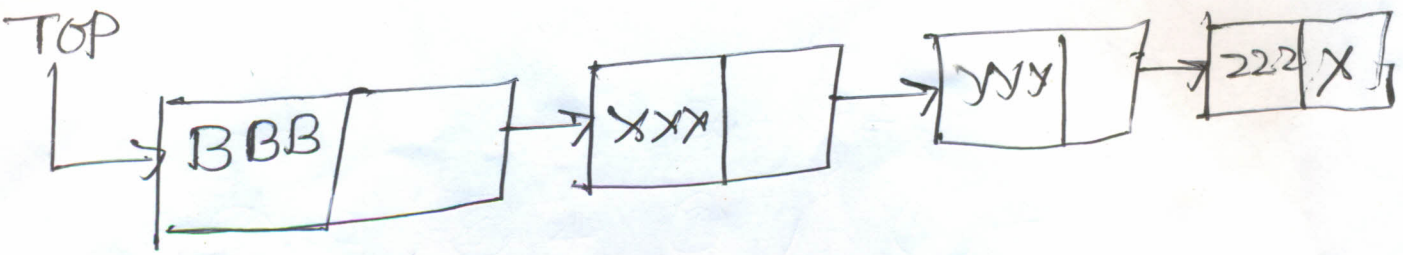
Fig. 6.29

The procedures which insert and delete elements in deques and the variations on those procedures are given as supplementary problem. As with queue, a complication may arise (a) when there is overflow, that is, when an element is to be inserted into a deque which is already full, or(b) when there is underflow, that is, when an element is to be deleted from a deque which is empty. The procedures must consider these possibilities.

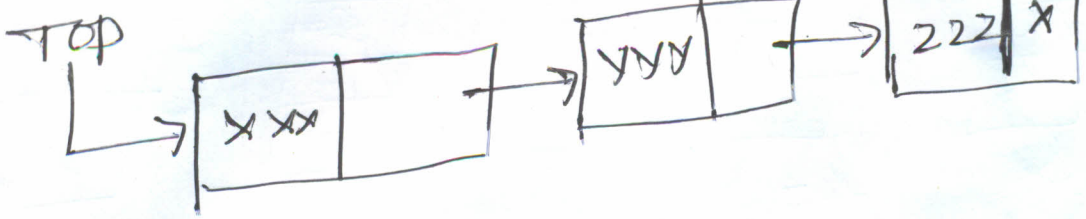
(4) consider the linked list given below and perform the following operations using diagram.



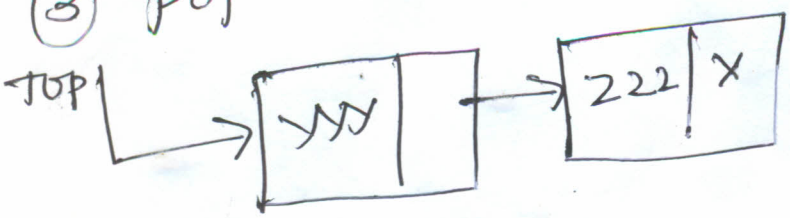
(1) PUSH BBB



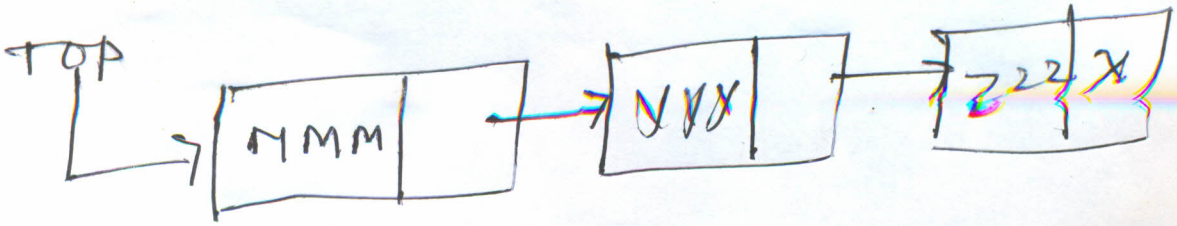
(2) POP

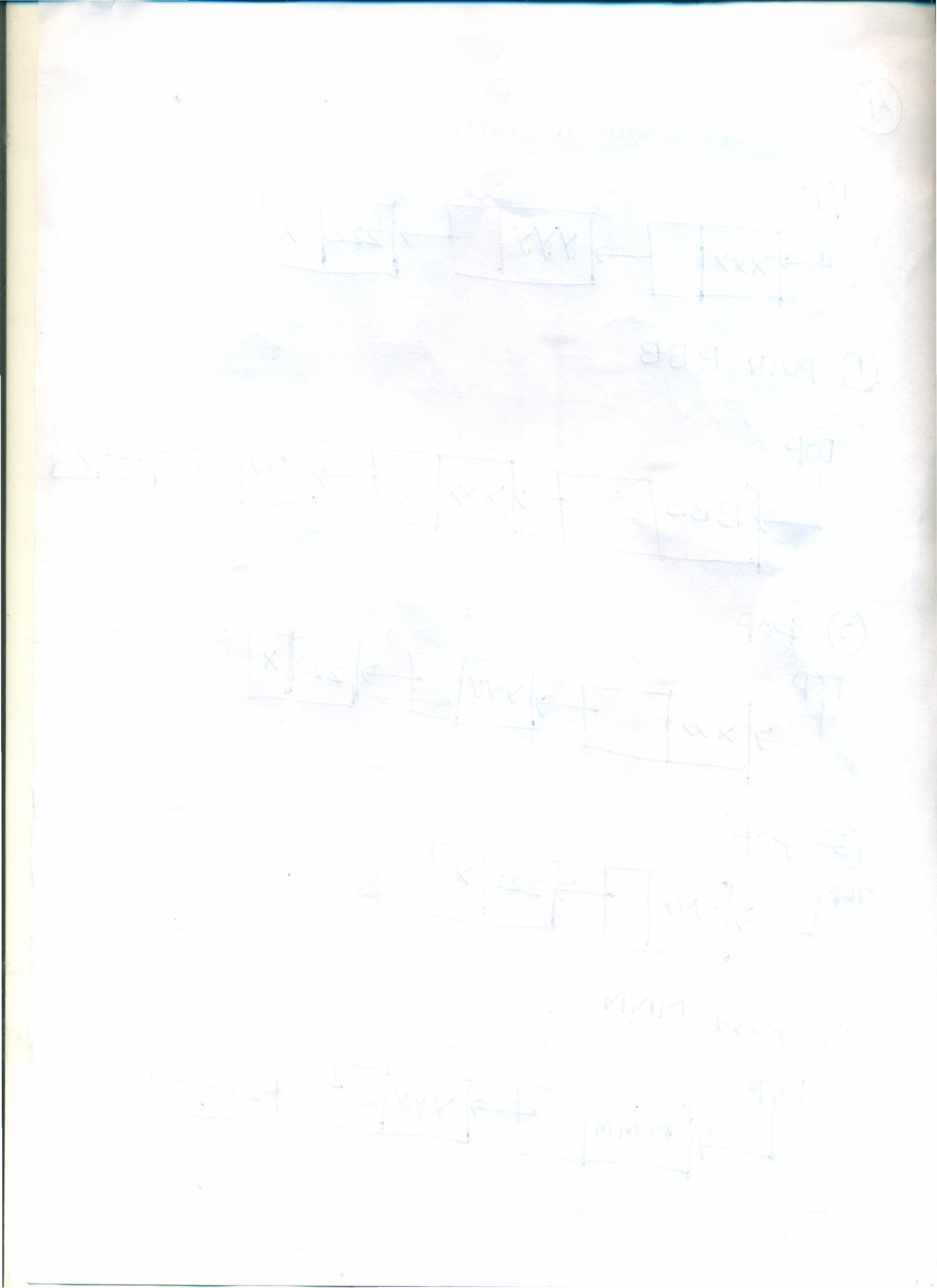


(3) POP



(4) PUSH MMM



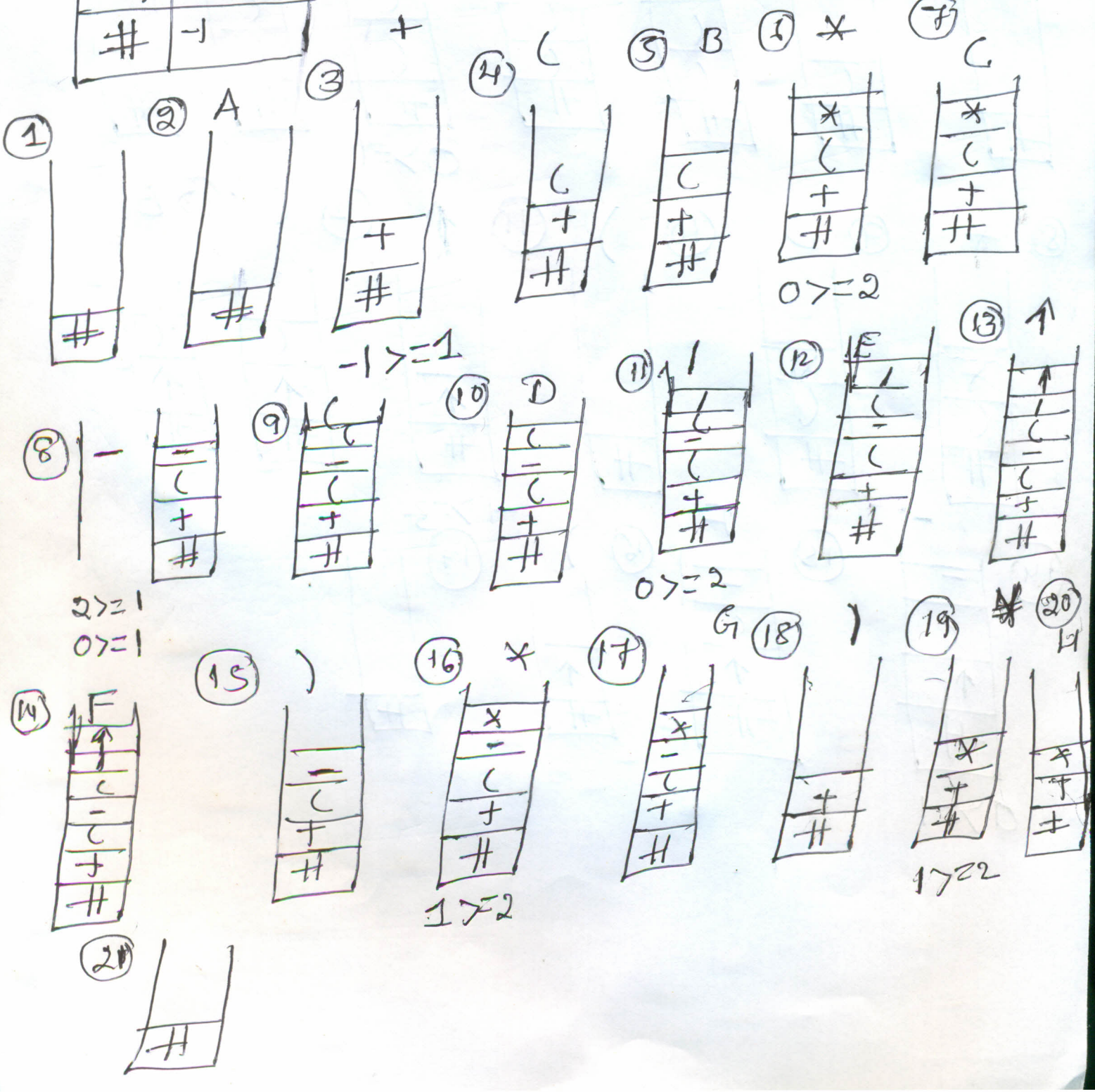


(7) convert the following infix into postfix expression using stack status.

Q. $A+(B*C-(D/E\uparrow F)*G)*H$

^	3
*/	2
+ -	1
( )	0
#	-1

$ABC*DEF\uparrow/G*-H*+$

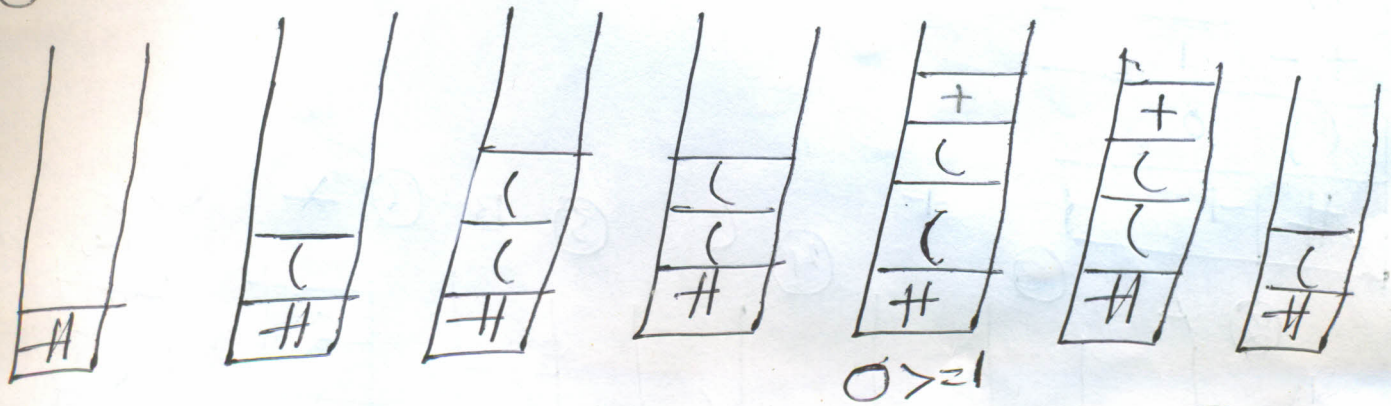


(8) Convert the following infix expression ~~into~~ postfix expression using stack status:

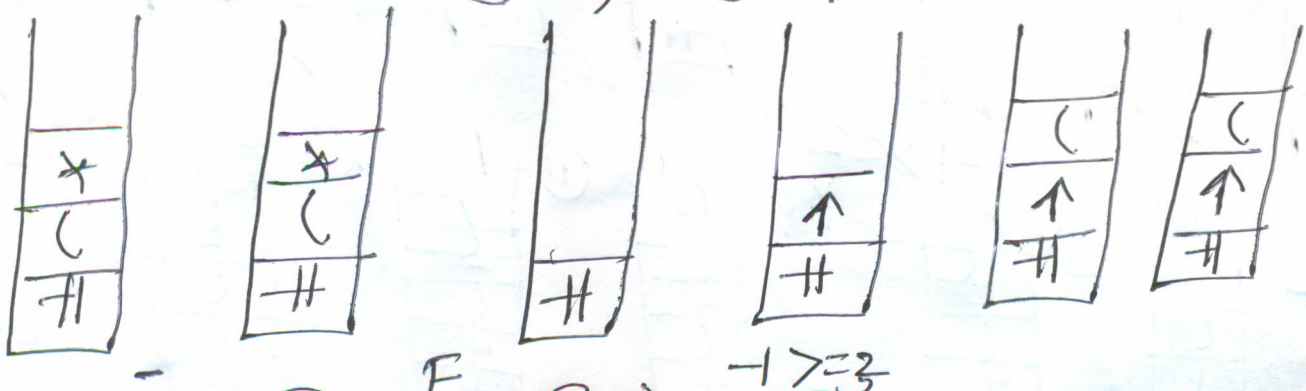
AB+D*EF-↑

((A+B)*D)↑(E-F)

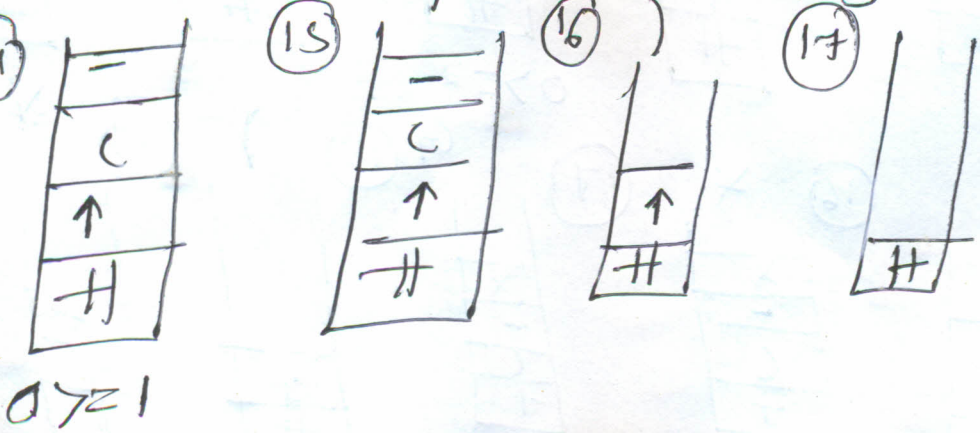
(1) (2) ((3) ((4) A (5) + (6) B (7))



(8) * (9) D (10)) (11) ↑ (12) ((13) E



(14) - (15) F (16)) (17)

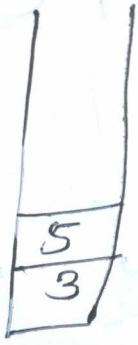


(9) Evaluate the following postfix expression showing stack status

P: 3, 5, +, 6, 4, −, *, 4, 1, −, 2, ↑, +

(1) 3

(2) 5

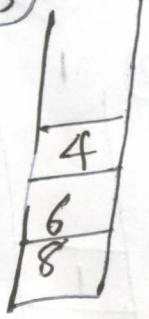


(3) +

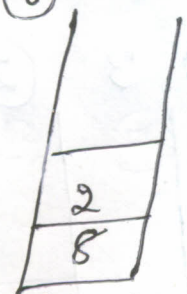
$op_2 = 5$
$op_1 = 3$
$Res = 3 + 5 = 8$

(4) 6

(5)

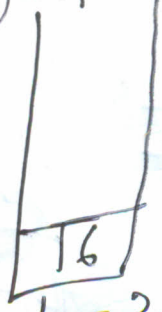


(6) −



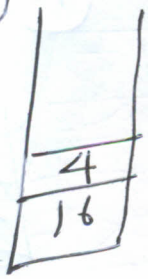
$op_2 = 4$
$op_1 = 6$
$Res = 6 - 4 = 2$

(7) *

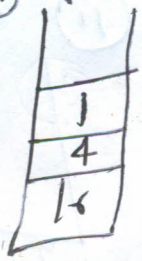


$op_2 = 2$
$op_1 = 8$
$Res = 8 \times 2 = 16$

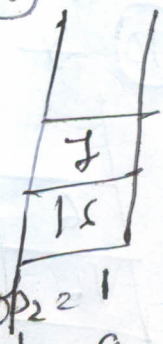
(8) 4



(9) 1

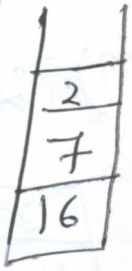


(10) −

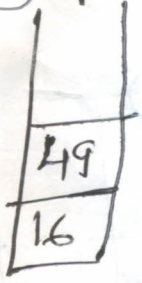


$op_2 = 1$
$op_1 = 8$
$Res = 8 - 1 = 7$

(11) 2

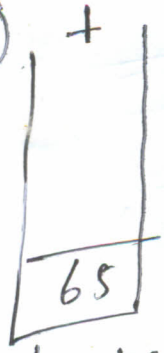


(12) ↑



$op_2 = 2$
$op_1 = 7$
$Res = 7^2 = 49$

(13) +



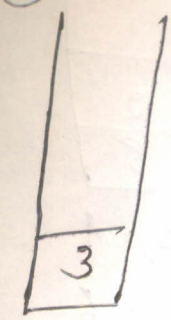
$op_2 = 49$
$op_1 = 16$
$Res = 16 + 49$
$= 65$

$Res = 65$

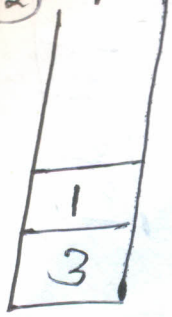
(10) Evaluate the following postfix expression showing the stack status.

P: 3, 1, +, 2, ↑, 7, 4, −, 2, *, +, 5, −

(1) 3



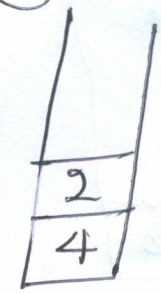
(2) 1



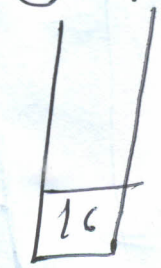
(3) +

$op_2 = 1$
$op_1 = 3$
$Re_1 = 3 + 1$
$= 4$

(4) 2



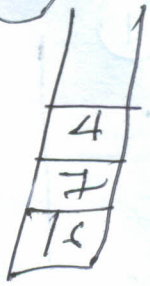
(5) ↑



$op_2 = 2$
$op_1 = 4$
$Re_1 = 4^2 = 16$

(6) 7

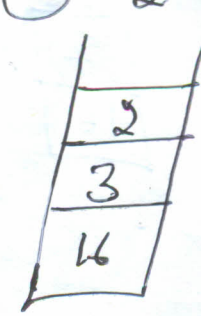
(7) 4



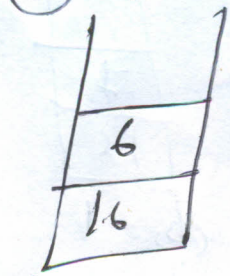
(8) −

$op_2 = 4$
$op_1 = 7$
$Re_1 = 7 - 4$
$= 3$

(9) 2

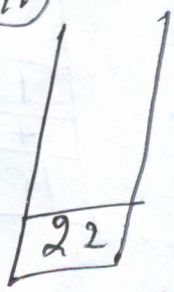


(10) *



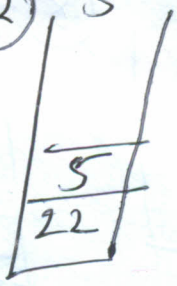
$op_2 = 2$
$op_1 = 3$
$Re_1 = 3 * 2 = 6$

(11) +

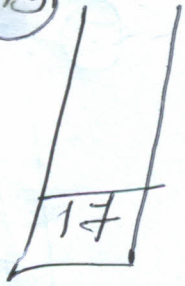


$op_2 = 6$
$op_1 = 16$
$Re_1 = 16 + 6$
$= 22$

(12) 5



(13) −



$op_2 = 5$
$op_1 = 22$
$Re_1 = 22 - 5$
$= 17$

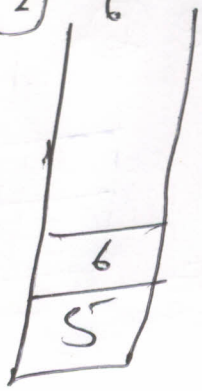
$Re_1 = 17$

Evaluate the following postfix expression showing the stack status.

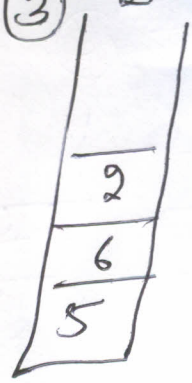
(11) P: 5, 6, 2, +, *, 1, 2, 4, /, -

(1) 5

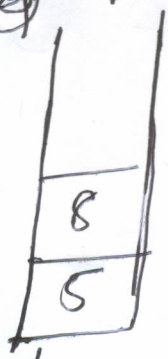
(2) 6



(3) 2

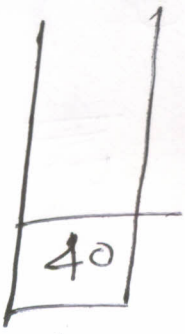


(4) +



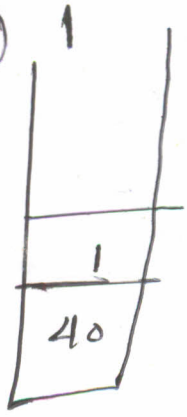
$op_2 = 2$
$op_1 = 6$
$R_4 = 6 + 2$
$= 8$

(5) *

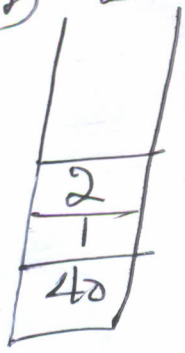


$op_2 = 8$
$op_1 = 5$
$R_4 = 5 * 8 = 40$

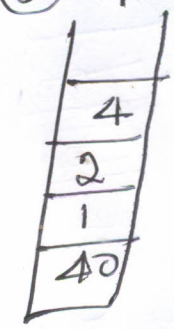
(6) 1



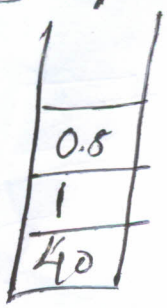
(7) 2



(8) 4

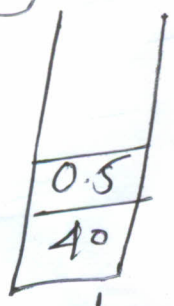


(9) /



$op_2 = 4$
$op_1 = 2$
$R_4 = 2/4 = 0.5$

(10) -



$op_2 = 0.5$
$op_1 = 1$
$R_4 = 1 - 0.5$
$= 0.5$

(9) Convert the given infix expression to prefix & postfix form.

a) $A*B+(C\wedge D/E/F)$

Post $\Rightarrow A*B+(CD\wedge/E/F)$

$\Rightarrow A*B+(CD\wedge E//F)$

$= \underline{AB*} + \underline{CD\wedge E//F/}$

$= AB* + CD\wedge E//F/$

$= \underline{AB*CD\wedge E//F/+}$

Pref $\Rightarrow A*B+(\wedge CD//E/F)$

$= A*B+(/\wedge CDE/F)$

$= A*B+//\wedge CDEF$

$= *AB+//\wedge CDEF$

$= \underline{+*AB//\wedge CDEF}$

(b) $a/b*c-d+e/f*(g+h)$

Post $\Rightarrow a/b*c-d+e/f*gh+$

$\Rightarrow ab/*c-d+e/f*gh+$

$= \underline{ab/c*} - d+ef/*gh+$

$= \underline{ab/c*} - d+\underline{ef/gh+*}$

$= \underline{ab/c*d} - \underline{+ef/gh+*}$

$= ab/c*d-ef/gh+*+$

Ans $\Rightarrow$ $a|b * c-d+e|f * +gh$

$= |ab * c-d+e|f * +gh$

$= *|abc-d + |ef * +gh.$

$= *|abc - d + *|ef+gh$

$= -*|abcd + *|ef+gh.$

$= +-*|abcd*|ef+gh$

(c) ~~$(x+y/z)$~~

$(x+y/z * w\wedge p) - R$

Sol : $(x+y/z * wp\wedge) - R$

$= (x+ yz| * wp\wedge) - R$

$= (x+ yz|wp\wedge|) - R$

$= xyz|wp\wedge|+ -R$

$= xyz|wp\wedge|+R-$

Ref: $= (x+y/z \times w \wedge p) - R$

$= (x+y/z \times \wedge w p) - R$

$= (x+/yz \times \wedge w p) - R$

$= (x+ \times /yz \wedge w p) - R$

$= + x \times /yz \wedge w p - R$

$= - + x \times /yz \wedge w p R$

(d) $(x+y \times z \wedge A/4) - p$

Ref: $= (x+y \times z A \wedge /4) - p$

$= (x+ yz A \wedge \times /4) - p$

$= (x+ yz A \wedge \times 4/) - p$

$= xyz A \wedge \times 4/ + - p$

$= xyz A \wedge \times 4/ + p -$

Post fix = $(x+y*z\wedge A/4) - P$

$= (x+y*\underline{\wedge zA}/4) - P$

$= (x+\underline{*y\wedge zA}/4) - P$

$= (x+/*y\wedge zA4) - P$

$= +x/*y\wedge zA4 - P$

$= -+x/*y\wedge zA4+P$

(e) $(A+B)*((C-D)/E*F$

Post fix $\Rightarrow$ $\underline{AB+}*CD-/E*F$

~~$= AB+CDD-*$~~

$= \underline{AB+CD-*}/E*F$

$= AB+CD-*E/*F$

$= AB+CD-*E^2/F*$

Prefix ⇒ $(A+B) * (C-D) / E+F$

$= +AB * -CD/E*F$

$= +*AB-CD/E*F$

$= /*+AB-CDE*F$

$= */*+AB-CDEF$

(f) $ab*c+(c/d\wedge e)-f$

Postfix $= ab*+(c/de\wedge)-f$

$= -ab*+cde\wedge/-f$

$= ab*cde\wedge/+-f$

$= ab*cde\wedge/+f-$

Prefix $\Rightarrow *ab+(c/\wedge de)-f$

$\Rightarrow *ab+/c\wedge de-f$

$= +*ab/c\wedge de-f$

$= -+*ab/c\wedge def$

(g) $(A-B*C\wedge D)\,|\,(E+F)$

Post $\Rightarrow (A-B*CD\wedge)\,|\,EF+$

$\Rightarrow (A-BCD\wedge*)\,|\,EF+$

$= ABCD\wedge*-\,|\,EF+$

$\Rightarrow ABCD\wedge*-EF+-$

Prefix $\Rightarrow (A-B*\wedge CD)\,|\,+EF$

$\Rightarrow (A-*B\wedge CD)\,|\,+EF$

$= -A*B\wedge CD\,|\,+EF$

$= |-A*B\wedge CD+EF$

(h) $(a|b)*(c*f+(a-d)*e)$

Post $\Rightarrow ab|*(c*f+ad-*e)$

$= ab|*(cf*+ad-e*)$

$= ab|*cf*ad-e*+$

$= ab|cf*ad-e*+*$

Prefix = (a/b) * (c * f + (a - d) * e)

= /ab * (c * f + -ad * e)

= /ab * (*cf + * -ade)

= /ab * + * cf * -ade

= * /ab + * cf * -ade

(1) A + B * (C - D * (E + F))

Postfix => A + B * (C - D * EF+)

= A + B * (C - DEF+ *)

= A + B * CDEF+ * -

= A + BCDEF+ * - *

= ABCDEF+ * - * +

Prefix => A + B * (C - D * + EF)

= A + B * (C - * D + EF)

= A + B * - C * D + EF

= A + * B - C * D + EF

= + A * B - C * D + EF

(j) a*b - c^d + e/f

Post = a*b - cd^ + e/f

= ab* - cd^ + ef/

= ab*cd^ - + ef/

= ab*dd^ - ef/+

Pref: *ab - ^cd + /ef

= -*ab^cd + /ef

= +-*db^cd/ef

BHANDARKARS' ARTS AND SCIENCE COLLEGE KUNDAPURA
COMPUTER SCIENCE DEPARTMENT
II – BCA DATA STRUCTURE -4 th unit (Q&A)

1. Write a note on i) Complete binary tree ii) Extended binary tree

i) Complete binary tree

- A binary tree with n nodes and of depth d is a strictly binary tree all of whose terminal nodes are at level d.
- In a complete binary tree. There is exactly one node at level 0, two nodes at level 1, and four nodes at level 2 and so on.
- If there are m nodes at Ievel **1** then a binary tree contains at most **2m** nodes at level 1 + 1.
- A binary tree has exactly one node at the root level and has at most 21 nodes at level 1. Taking into consideration of this property, we can show further that a complete binary tree of depth d contains exactly 21 nodes at each level 1. The value 1 ranges from 0 to d.

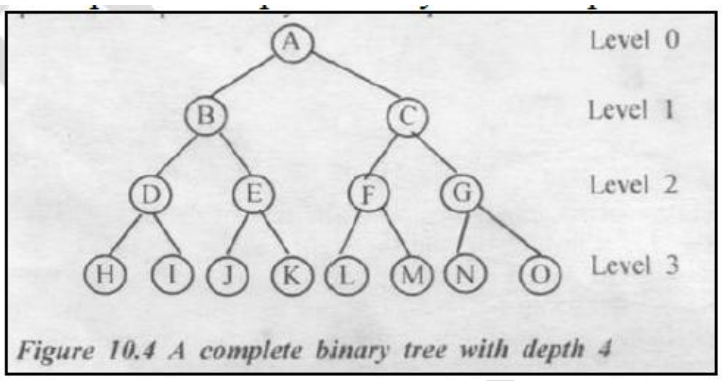
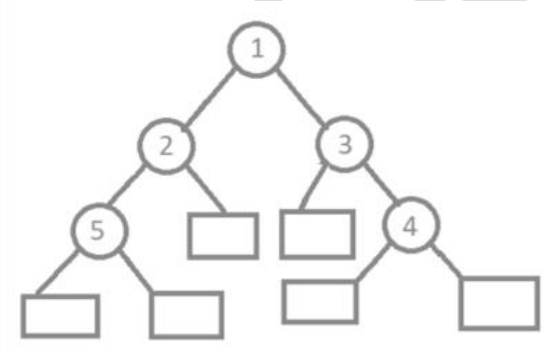


Figure 10.4 A complete binary tree with depth 4

ii) Extended binary tree

Extended binary tree is a type of binary tree in which all the null sub tree of the original tree are replaced with special nodes called **external nodes** whereas other nodes are called **internal nodes**



Extended Binary Tree

Properties of External binary tree

1. The nodes from the original tree are internal nodes and the special nodes are external nodes.
2. All external nodes are leaf nodes and the internal nodes are non-leaf nodes.
3. Every internal node has exactly two children and every external node is a leaf. It displays the result which is a complete binary tree.

2. With an example, explain linked representation of binary tree.

Linked Representation of Binary Trees

Consider a binary tree T. Unless otherwise stated or implied, T will be maintained in memory by mean linked representation which uses three parallel arrays, INFO, LEFT and RIGHT, and a pointer variable ROOT as follows.

First of all, each node of T will correspond to a location K such that:

1.INFO[K] contains the data at the node

2.LEFT[K] contains the location of the left child of node N. ,

3.RIGHT[K] contains the location of the right child of node N.

Furthermore, ROOT will contain the location of the root R of T. If any subtree is empty, then the responding pointer will contain the null value; if the tree T itself is empty, then ROOT will contain the null value.

In some applications, it is necessary to include the father (or parent) field. In such situation one more field to represent the father of a node is inserted into the structure definition of a binary tree node. That is

```
struct node
 {
char data;
char father;
struct node *lchild;
struct node *rchild;
};
 typedef struct node BTNODE;
```

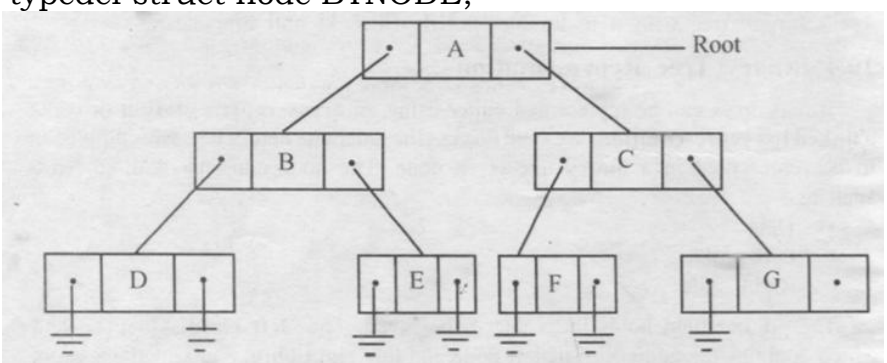


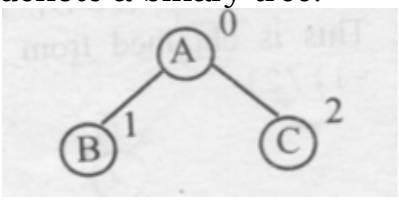
Figure 10.6 A binary tree in linked representation

A binary tree contain one root node and some non-terminal and terminal nodes(leaves). It is clear from the observation of a binary tree that the non-terminal nodes have their left child and right child nodes. But, the terminal nodes have no left child and right child nodes. Their lchild and rchild pointer are set to NULL. Here, the non-terminal nodes are called internal nodes and terminal nodes are called external nodes. In a specific application user may maintain two sets of nodes for both internal nodes and external nodes.

With an example, explain sequential representation of binary tree.

An array can be used to store the nodes of a binary tree. The nodes stored in an array are accessible sequentially. In C, arrays start with index 0 to (MAXSIZE - 1). Here, numbering of binary tree nodes starts from 0 rather than 1. The maximum number of nodes is specified by MAXSIZE.

The root node is always at index 0. Then, in successive memory locations the left child and right child are stored. Consider a binary tree with only three nodes as shown. Let BT denote a binary tree.



The array representation of this binary tree is as follows. Here, A is the father of B and C. B is the left child of A and C is the right child of A. Let us extend the above tree by one more level as shown below.

	BT
BT[0]	A
BT[1]	B
BT[2]	C

With an example, explain the 2 methods of tree representation in memory.
Linked representation
Sequential representation

3. What are the 3 standard ways of traversing a tree T with root R. write steps of each

traversal using recursion.
There are three popular ways of binary tree traversal. They are,
1. Preorder traversal
2. Inorder traversal
3. Postorder traversal

Preorder Traversal

The preorder traversal of a non-empty binary tree is defined as follows.
1. Visit the root node.
2. Traverse the left subtree in preorder.
3. Traverse the right subtree in preorder.

```
preorder(BTNODE *tree)
 {
if(tree != NULL)
{
printf ("%c“, tree->data); /*step 1*/
preorder(tree -> lchild); /*step 2*/
preorder(tree -> rchild); /*step 3*/
}
return;
}
```

Inorder Traversal

The **inorder** traversal of a non-empty binary tree is defined as follows.
1. Traverse the left subtree in inorder.
2. Visit the root node.
3. Traverse the right subtree in inorder.

```
inorder(BTNODE *tree)
{
```

```
if(tree != NULL)
{
 inorder(tree -> lchild); /*step 1*/
printf ("%c“, tree->data); /*step 2*/
inorder(tree -> rchild); /*step 3*/
}
return;
}
```

Postorder Traversal

The postorder traversal of a non-empty binary tree IS defined as follows.

1. Traverse the left subtree in postorder.

2. Traverse the right subtree in postorder.

3. Visit the root node.

```
postorder(BTNODE *tree)
 {
if(tree != NULL)
{
postorder(tree -> lchild); /*step 1*/
postorder(tree -> rchild); /*step 2*/
printf ("%c“, tree->data); /*step 3*/
}
return;
}
```

4. Explain the method of representing the graphs using sequential method with an example.

Adjacency Matrix Suppose G is a simple directed graph with m nodes, and suppose the nodes of G have been ordered and are called v1, v2,…., vm. Then the adjacency matrix A = (aij) of the graph G is the m x m matrix defined as follow :

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \text{ that is, if there is an edge } (v_i, v_j) \\ 0 & \text{otherwise} \end{cases}$$

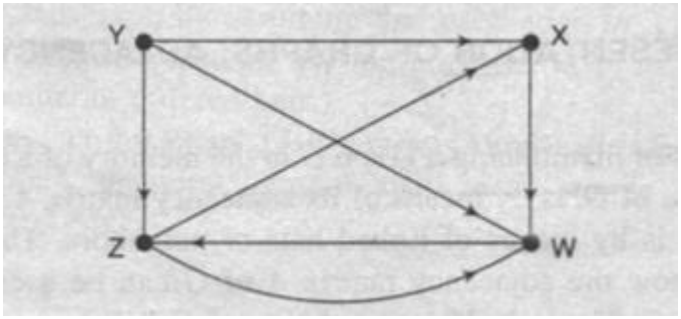
Such a matrix A, which contains entries of only 0 and 1, is called a **bit matrix** or a **Boolean matrix**.

The adjacency matrix A of the graph G does depend on the ordering of the nodes of G; that is, a different ordering of the nodes may result in a different adjacency matrix.

Consider the graph G in Fig. Suppose the nodes are stored in memory in a linear array DATA as follows: *DATA: X, Y, Z, W* Then we assume that the ordering of the nodes in G is as follows: v1 = X, v2 = Y, v3 = Z and v4 = W. The adjacency matrix *A* of *G* is as follows:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Note that the number of 1's in A is equal to the number of edges in G.



Path Matrix

Let G be a simple directed graph with m nodes, v1, v2, ... , vm. The path matrix or reachability matrix of G is the m-square matrix P = (Pij) defined as follows:

$$P_{ij} = \begin{cases} 1 & \text{if there is a path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Suppose there is a path from vi to vj. Then there must be a simple path from vi to vj when vi ≠ vj , or there must be a cycle from vi to vj when vi = vj.

Consider the graph G with m = 4 nodes. Adding the matrices A, A2, A3 and A4, we obtain the following matrix B4, and, replacing the nonzero
entries in B4 by 1, we obtain the path matrix P of the graph G:

$$B_4 = \begin{pmatrix} 1 & 0 & 2 & 3 \\ 5 & 0 & 6 & 8 \\ 3 & 0 & 3 & 5 \\ 2 & 0 & 3 & 3 \end{pmatrix} \quad \text{and} \quad P = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Examining the matrix *P*, we see that the node *v2* is not reachable from any of the other nodes. Recall that a directed graph *G* is said to be ***strongly connected*** if, for any pair of nodes *u* and *v* in *G*, there are both a path from *u* to *v* and a path from *v* to *u* Accordingly, *G* is strongly connected if and only if the path matrix *P* of *G* has no zero entries. Thus the graph *G* in the previous Fig. is not strongly connected.

5. What is adjacency matrix and path matrix, explain with an example.

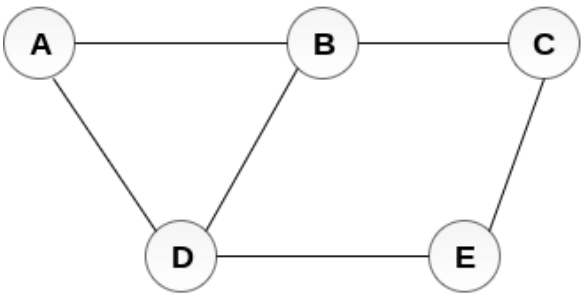
Refer question number 4.

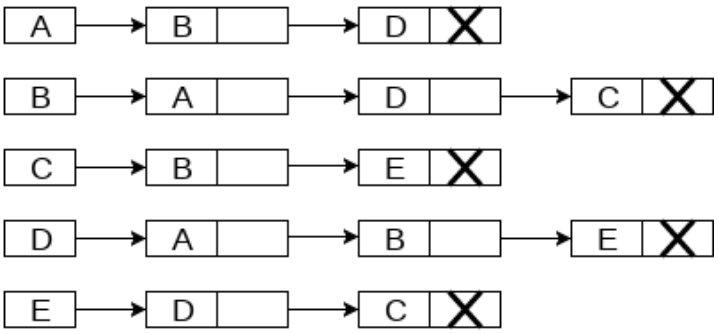
6. Explain linked representation of the graph with an example.

Linked Representation

In the linked representation, an adjacency list is used to store the Graph into the computer's memory.

Consider the undirected graph shown in the following figure and check the adjacency list representation.





Undirected Graph **Adjacency List**

An adjacency list is maintained for each node present in the graph which stores the node value and a pointer to the next adjacent node to the respective node. If all the adjacent nodes are traversed then store the NULL in the pointer field of last node of the list. The sum of the lengths of adjacency lists is equal to the twice of the number of edges present in an undirected graph.

7. What are the 4 steps involved in the deletion of a node N in the graph G.

1.Find the location LOC of the node N in G.
2.Delete all edges ending at N;that is,delete LOC from the list of successors of each node M in G.(This step requires traversing the node list of G.)
3.Delete all the edges beginning at N.This is accomplished by finding the location BEG of the first successor and the location END of the last successor of N,and then adding the successor list of N to the free AVAIL list.
4.Delete N itself from the list NODE.

8. Write an algorithm for breadth first search (BFS) for a graph.

This algorithm executes a breadth-first search on a graph *G* beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS = 1).
2. Put the starting node A in QUEUE and change its status to the waiting state (STATUS = 2). 3. Repeat Steps 4 and 5 until QUEUE is empty:
4. Remove the front node N of QUEUE. Process N and change the status of N to the processed state (STATUS = 3).
5. Add to the rear of QUEUE all the neighbors of N that are in the steady state (STATUS = 1), and change their status to the waiting state (STATUS = 2).
[End of Step 3 loop.]
6. Exit.

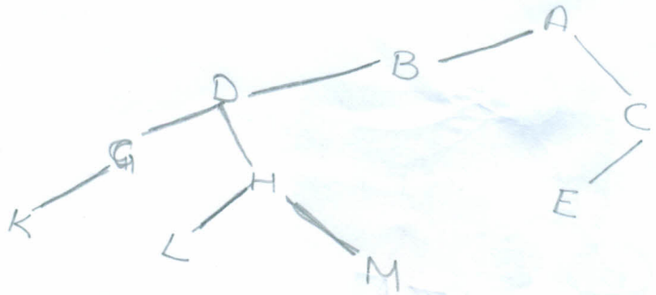
9. Write an algorithm for depth first search (DFS) for a graph.

Algorithm:

1. Initialize all nodes to the ready state (STATUS = 1).
2. Push the starting node A onto STACK and change its status to the waiting state(STATUS= 2).
3. Repeat Steps 4 and 5 until STACK is empty.
4. Pop the top node N of STACK. Process N and change its status to the processed state (STATUS = 3).

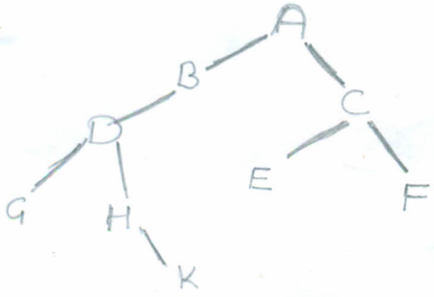
5. Push onto STACK all the neighbors of N that are still in the ready state (STATUS = 1), and change their status to the waiting state (STATUS = 2).
[End of Step 3 loop.]
6. Exit.

(7) write the preorder, inorder & postorder traversal for the given tree.



Preorder: ABDGKHLMCE
Inorder: KGDLHMBAEC
Postorder: KGLMHDBECA

(8) write the preorder, inorder and postorder traversal for the given tree.

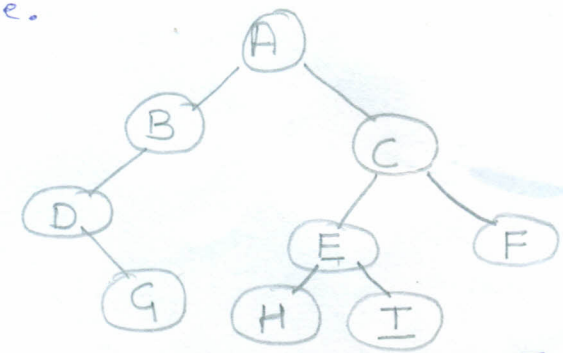


Preorder: ABDGHKCEF
Inorder: GDHKBAECF
Postorder: GKHDBEFCA

(9) write the preorder, inorder and post order traversal for the given tree.

Preorder: ABMFZOLWXYCN
Inorder: ZFOMBLXWYACN
Postorder: ZOFMXYWLBNCA

(10) write the preorder, inorder and postorder traversal for the given tree.

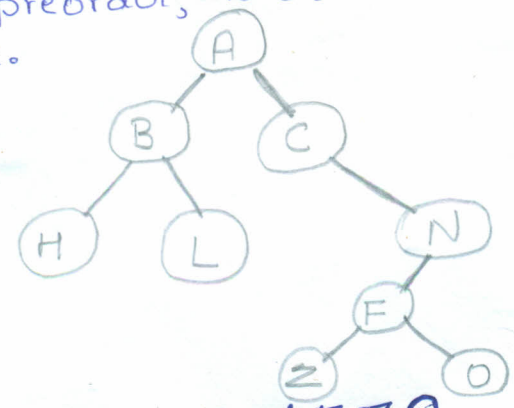


Preorder: A B D G C E H I F

Inorder : D G B A H E I C F

Postorder : G D B H I E F C A

(11) write the preorder, inorder and postorder traversal for the given tree.

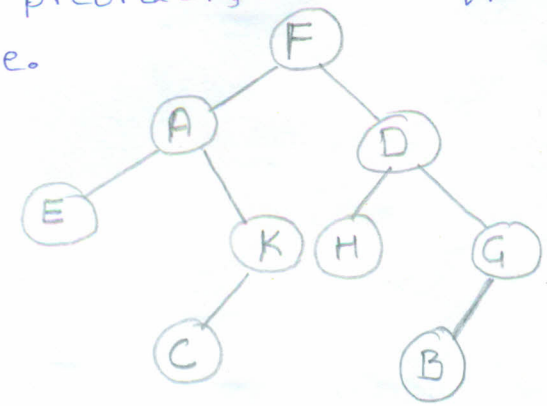


Preorder: A B H L C N F Z O

Inorder: H B L A C Z F O N

Postorder: H L B Z O F N C A

(12) write the preorder, inorder & postorder traversal for the given tree.

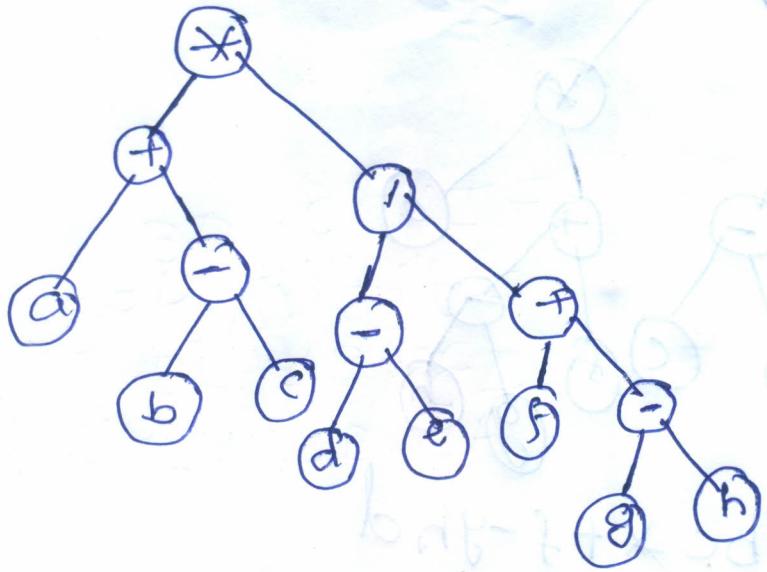


Preorder: F A E K C D H G B

Inorder: E A C K F H D B G

Postorder: E C K A H B G D F

(13) Draw the binary tree for the given algebraic expression: $[a+(b-c)]*[(d-e)/(f+g-h)]$. Also write preorder, inorder & postorder traversal methods for the tree.

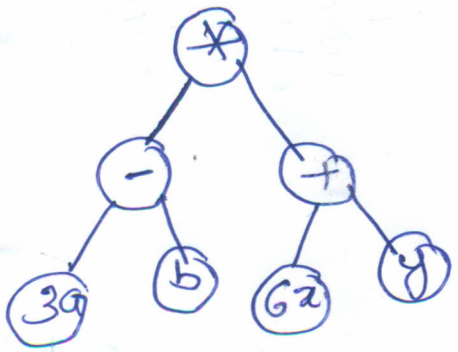


Preorder: * + a - b c / - d e + f - g h

Inorder: a + b - c * d - e / f + g - h

Postorder: a b c - + d e - f g h - + / *

(14) construct tree for the given infix expression: $(3a-b)(6x+y)$. Also write preorder, inorder & postorder traversal methods for the tree.

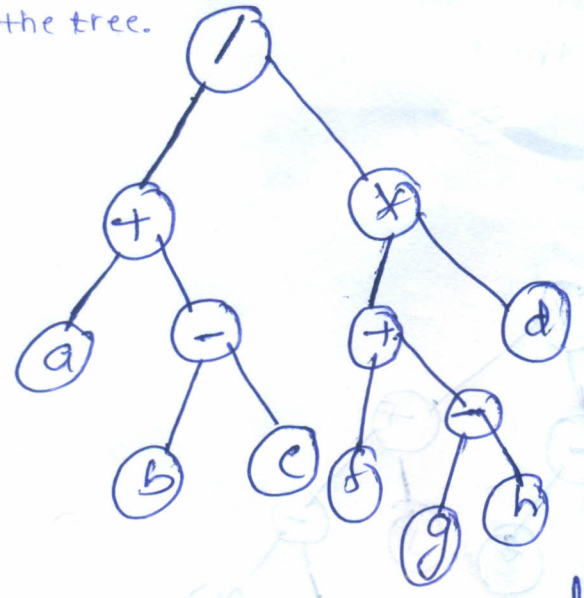


Preorder: * - 3a b + 6x y

Inorder: 3a - b * 6x + y

Postorder: 3a b - 6x y + *

(15) Draw the binary tree for the given algebric expression: (a+b-c)/((f+g-h)*d). Also write preorder, inorder & postorder traversal methods for the tree.

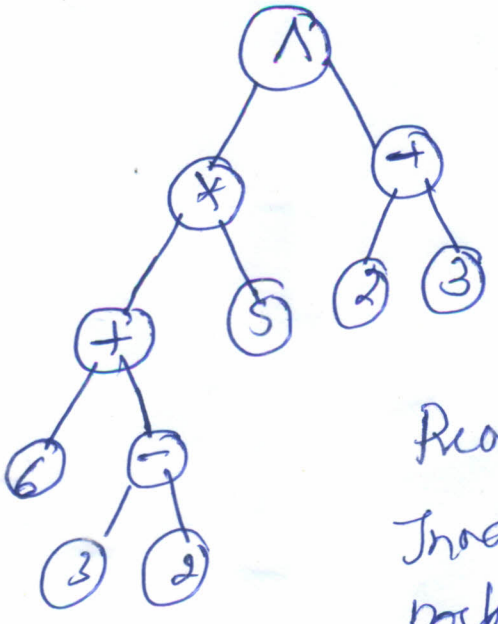


Preorder: /+a-bc*+f-ghd

Inorder: a+b-c/f+g-h*d

postorder: abc-+fgh-+d*/

(16) Draw the binary tree for the given algebric expression: ((6+(3-2)*5)^2+3). Also write preorder, inorder & postorder traversal methods for the tree.



Preorder: ^*+6-325+23

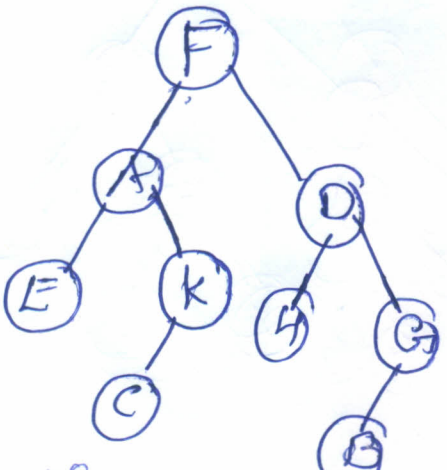
Inorder: 6+3-2*5^2+3

postorder: 632-+5*23+^

(17) Draw the binary tree for the following

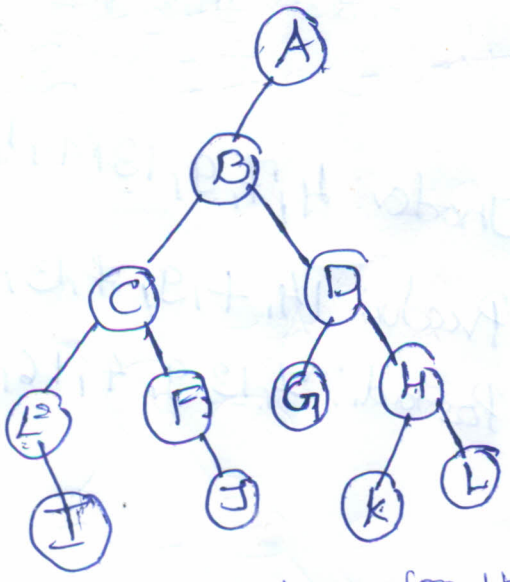
INORDER : E, A, C, K, F, H, D, B, G

PREORDER : F, A, E, K, C, D, H, G, B



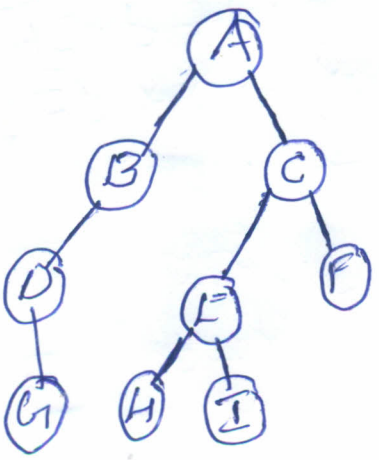
(18) Draw the binary tree for the following preorder, inorder & traversal. PREORDER : A, B, C, E, I, F, J, D, G, H, K, L .

INORDER : E, I, C, F, J, B, G, D, K, H, L, A .



(19) Draw the binary tree for the following postorder & inorder traversal. POSTORDER : G, D, B, H, I, E, F, C, A .

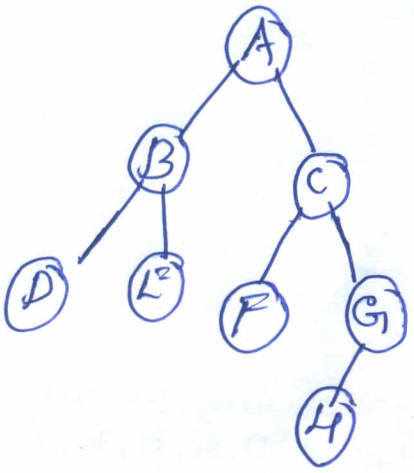
INORDER : D, G, B, A, H, E, I, C, F .



(19) Draw the binary tree for the following inorder and postorder traversal.

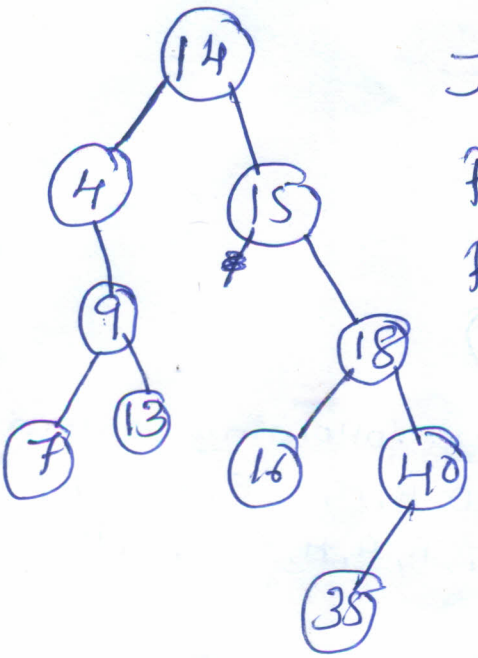
INORDER : D, B, E, A, F, C, H, G.

POSTORDER : D, E, B, F, H, G, C, A.



(21) Draw a binary search tree for the following list of numbers and traverse it in preorder, inorder and postorder:

14, 15, 4, 9, 7, 18, 40, 35, 16, 13.

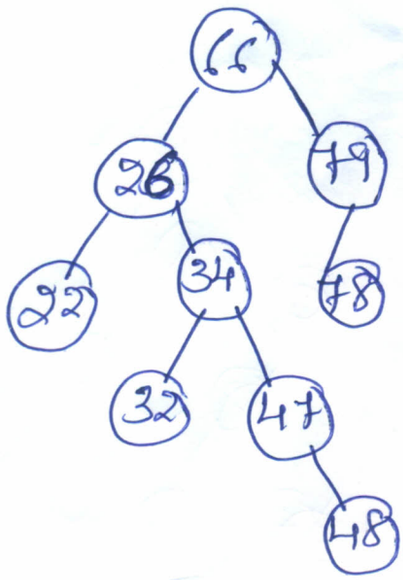


Inorder: 4, 7, 9, 13, 14, 15, 16, 18, 35, 40

Preorder: 14, 4, 9, 7, 13, 15, 18, 16, 40, 35

Postorder: 7, 13, 9, 4, 16, 35, 40, 18, 15, 14

(22) Construct a binary search tree for the following list of numbers & traverse it in preorder, inorder and postorder: 66, 26, 22, 34, 47, 79, 48, 32, 78.

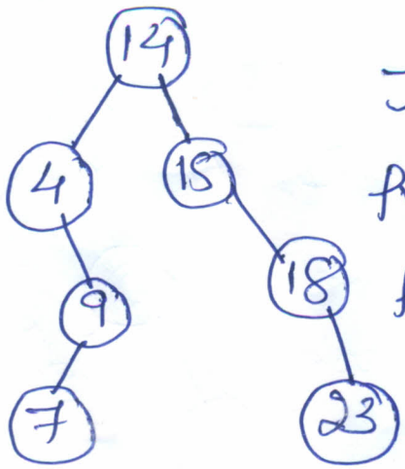


Inodr: 22, 26, 32, 34, 47, 48, 66, 78, 79

Prodr: 66, 26, 22, 34, 32, 47, 48, 79, 78

Postor: 22, 32, 48, 47, 34, 26, 78, 79, 66

(23) Construct a binary search tree for the following list of numbers and traverse it in preorder, inorder & postorder ~~and~~ : 14, 15, 4, 9, 7, 18, 23.



Inodr: 4, 7, 9, 14, 15, 18, 23

Prodr: 14, 4, 9, 7, 15, 18, 23

Postor: 7, 9, 4, 23, 18, 15, 14.