FACULTY OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CYCLE TEST – II

ACADEMIC YEAR 2019 – 2020 / ODD

BATCH – 1 – ANSWER KEY

Program Offered: B. Tech / CSE | **Year / Semester: II / III**
Max. Marks: 50 | **Duration: 2 periods**
Date of Exam: 20/09/2019

Course Code & Title: 18CSS201J – DATA STRUCTURES AND ALGORITHMS

PART –A

ANSWER ALL QUESTIONS (10*1=10 Marks)

1. A queue follows ________
   a. FIFO (First In First Out) principle
   b. LIFO (Last In First Out) principle
   c. Ordered array
   d. Linear tree
2. In a stack, if a user tries to remove an element from empty stack it is called ________
   a. Underflow
   b. Empty collection
   c. Overflow
   d. Garbage Collection
3. An array elements are always stored in ____________ memory locations.
   a. Sequential
   b. Random
   c. Sequential and random
   d. None of the above.
4. If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time, in what order will they be removed?
   a. ABCD
   b. DCBA
   c. DCAB

d. ABDC

5. In the array implementation of circular queue, which of the following operation take worst case linear time?

   a. Insertion

   b. Deletion

   c. To empty a queue

   d. None

6. Reverse polish Notation is the other name of

   a. Infix expression

   b. Prefix expression

   c. postfix expression

   d. Algebraic expression

7. Which type of linked list contains a pointer to the next as well as the previous node in the sequence?

   a. Singly linked list

   b. Doubly linked list

   c. Circular linked list

   d. Circular header linked list.

8. The circular queue will be full only when

   A) FRONT= MAX-1 and REAR=MAX-1
   B) FRONT=0 and REAR=MAX-1
   C) FRONT=MAX-1 and REAR=0
   D) FRONT=0 and REAR=0

   a. A and C
   b. C and D
   c. A and B
   d. B and D

9. In input restricted queue, the insertions can be done **ANSWER: A(1 mark to be given)**

   A) Only in one end
   B) Both the ends
   C) Deletion in one end
   D) Deletion in both the ends

   a. A and D
   b. A and B
   c. B and D
   d. C and D

10. In a circular linked list

   a. Components are all linked together in some sequential manner.
   b. There is no beginning and no end.
   c. Components are arranged hierarchically.
   d. Forward and backward traversal within the list is permitted.

PART –B

ANSWER ANY FIVE QUESTIONS (5*4=20 Marks)

11. Storing of sparse matrices need extra consideration – Justify. How are sparse matrices stored efficiently in computer's memory?

- Sparse matrix is a matrix that has large number of elements with a zero value. In order to efficiently utilize the memory, specialized algorithms and data structures that take advantage of the sparse structure should be used. If we apply the operations using standard matrix structures and algorithms to sparse matrices, then the execution will slow down and the matrix will consume large amount of memory. Sparse data can be easily compressed, which in turn can significantly reduce memory usage.
- Types of sparse matrix: Lower-triangular matrix, Upper-triangular matrix, Tri-diagonal matrix.
- To store a sparse matrix efficiently in the memory, we can use a one-dimensional array which stores only non-zero elements. The mapping between a two-dimensional matrix and a one-dimensional array can be done in any one of the following ways: Row-wise mapping, Column-wise mapping, Diagonal-wise mapping.

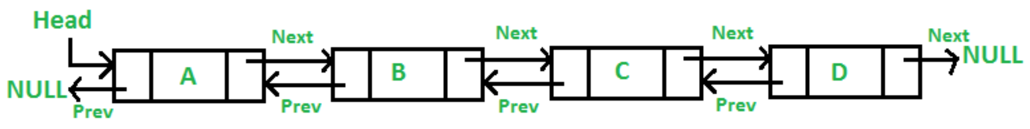
12. Make a comparison between a linked list and a linear array. Which one will you prefer and when?

BASIS FOR COMPARISON	ARRAY	LINKED LIST
Basic	It is a consistent set of a fixed number of data items.	It is an ordered set comprising a variable number of data items.
Size	Specified during declaration.	No need to specify; grow and shrink during execution.
Storage Allocation	Element location is allocated during compile time.	Element position is assigned during run time.
Order of the elements	Stored consecutively	Stored randomly
Accessing the element	Direct or randomly accessed, i.e., Specify the array index or subscript.	Sequentially accessed, i.e., Traverse starting from the first node in the list by the pointer.
Insertion and deletion of element	Slow relatively as shifting is required.	Easier, fast and efficient.
Searching	Binary search and linear search	linear search
Memory required	Less	More
Memory Utilization	Ineffective	Efficient

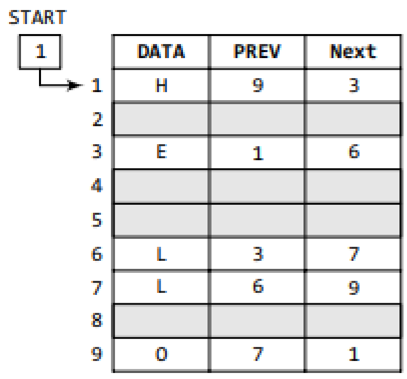
- Arrays are preferred over linked list when the size of data is known in prior.
- Each data structure has strengths and weaknesses which affect performance depending on the task. Arrays allow random access and require less memory per element (do not need space for pointers) while lacking efficiency for insertion/deletion operations and memory allocation. On the contrary, linked lists are dynamic and have faster insertion/deletion time complexities. However, linked list have a slower search time and pointers require additional

memory per element in the list.

13. Give the memory representation of a double linked list with example.



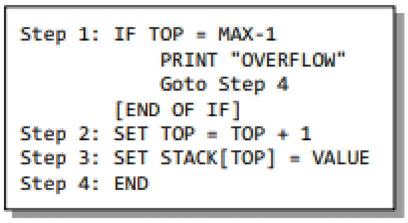
```
/* Node of a doubly linked list */
struct Node {
   int data;
   struct Node* next; // Pointer to next node in DLL
   struct Node* prev; // Pointer to previous node in DLL
};
```



START 1

	DATA	PREV	Next
1	H	9	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	0	7	1

14. Develop an algorithm to insert an element in a stack using arrays.



```
Step 1: IF TOP = MAX-1
              PRINT "OVERFLOW"
              Goto Step 4
        [END OF IF]
Step 2: SET TOP = TOP + 1
Step 3: SET STACK[TOP] = VALUE
Step 4: END
```

15. Differentiate between an iterative function and a recursive function. Which one will you prefer to use and in what circumstances?

- Recursion is more of a top-down approach to problem solving in which the original problem is divided into smaller sub-problems. On the contrary, iteration follows a bottom-up approach that begins with what is known and then constructing the solution step by step.
- Recursion is an excellent way of solving complex problems especially when the problem can be defined in recursive terms. For such problems, a recursive code can be written and modified in a much simpler and clearer manner. In some cases, recursive programs may require substantial amount of run-time overhead.
- One must use recursion only to find solution to a problem for which no obvious iterative solution is known.

16. Discuss the general rule of processing the elements of a priority queue.

- An element with higher priority is processed before an element with a lower priority.
- Two elements with the same priority are processed on a first-come-first-served (FCFS) basis.

17. Consider the queue given below which has FRONT = 1 and REAR = 5. Now perform the following operations on the queue:

	A	B	C	D	E				

	A	B	C	D	E	F			

a) Add F

b) Delete two letters

			C	D	E	F			

c) Add G

			C	D	E	F	G		

d) Add H

			C	D	E	F	G	H	

e) Delete four letters

							G	H	

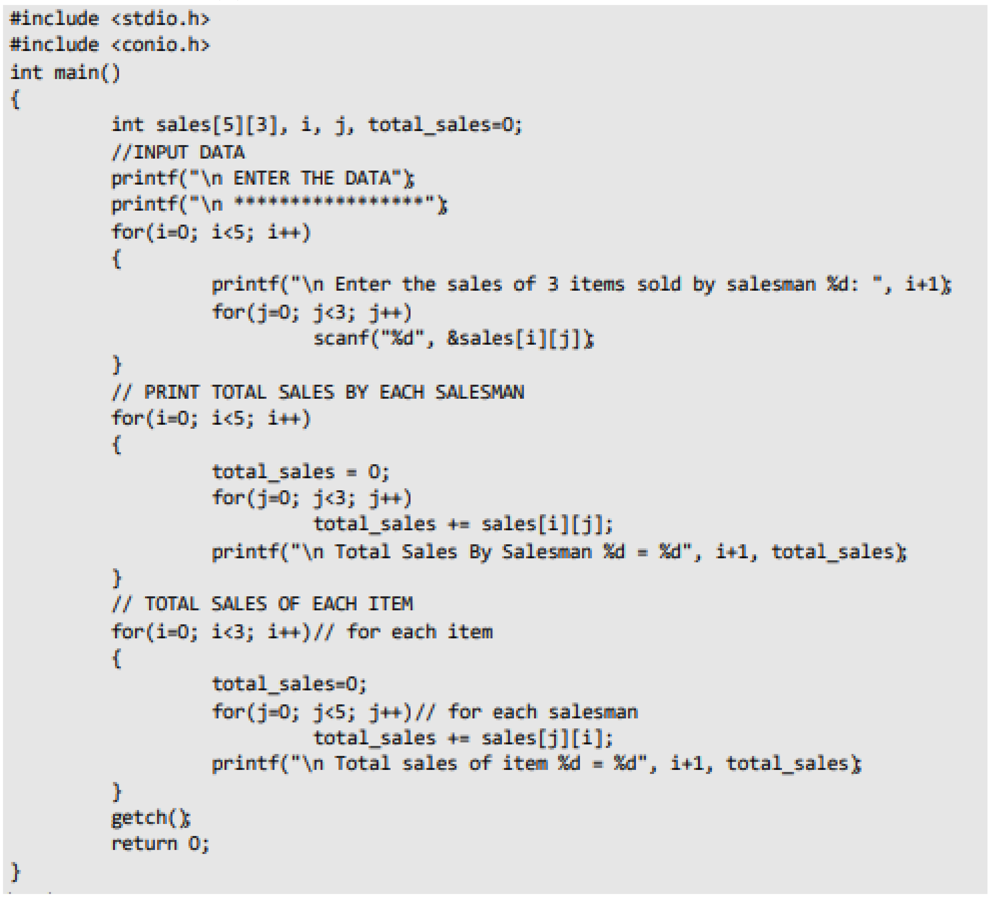
f) Add I

							G	H	I

PART – C

ANSWER ANY TWO QUESTION (2*10 = 20 Marks)

18a. In a small company there are five salesmen. Each salesman is supposed to sell three products. Write a program using a 2D array to print (i) the total sales by each salesman and (ii) total sales of each item.



```
#include <stdio.h>
#include <conio.h>
int main()
{
        int sales[5][3], i, j, total_sales=0;
        //INPUT DATA
        printf("\n ENTER THE DATA");
        printf("\n *****************");
        for(i=0; i<5; i++)
        {
                printf("\n Enter the sales of 3 items sold by salesman %d: ", i+1);
                for(j=0; j<3; j++)
                        scanf("%d", &sales[i][j]);
        }
        // PRINT TOTAL SALES BY EACH SALESMAN
        for(i=0; i<5; i++)
        {
                total_sales = 0;
                for(j=0; j<3; j++)
                        total_sales += sales[i][j];
                printf("\n Total Sales By Salesman %d = %d", i+1, total_sales);
        }
        // TOTAL SALES OF EACH ITEM
        for(i=0; i<3; i++)// for each item
        {
                total_sales=0;
                for(j=0; j<5; j++)// for each salesman
                        total_sales += sales[j][i];
                printf("\n Total sales of item %d = %d", i+1, total_sales);
        }
        getch();
        return 0;
}
```

Output:

```
ENTER THE DATA
Enter the sales of 3 items sold by salesman 1: 23 23 45
Enter the sales of 3 items sold by salesman 2: 34 45 63
Enter the sales of 3 items sold by salesman 3: 36 33 43
Enter the sales of 3 items sold by salesman 4: 33 52 35
Enter the sales of 3 items sold by salesman 5: 32 45 64
Total Sales by Salesman 1 = 91
Total Sales by Salesman 2 = 142
```

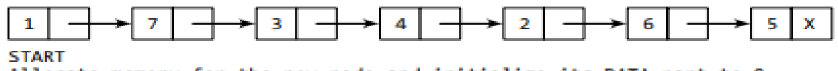
Total Sales by Salesman 3 = 112
Total Sales by Salesman 4 = 120
Total Sales by Salesman 5 = 141
Total sales of item 1 = 158
Total sales of item 2 = 198
Total sales of item 3 = 250

18b. Discuss the following operations of single linked list with example:

A) **Inserting a node after a given node – 3.5 marks**

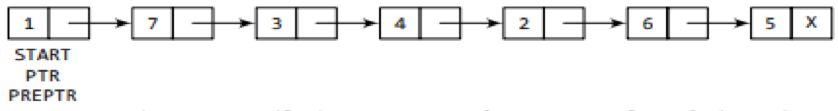
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA
         != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step 12: EXIT
```

In Step 5, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list. Then we take another pointer variable PREPTR which will be used to store the address of the node preceding PTR. Initially, PREPTR is initialized to PTR. So now, PTR, PREPTR, and START are all pointing to the first node of the linked list. In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM. We need to reach this node because the new node will be inserted after this node. Once we reach this node, in Steps 10 and 11, we change the NEXT pointers in such a way that new node is inserted after the desired node.

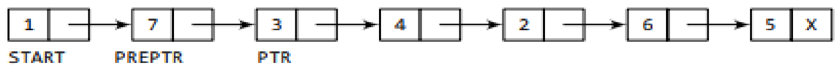


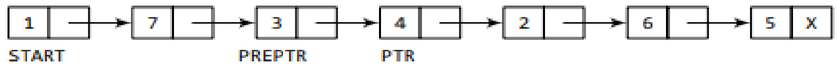
Allocate memory for the new node and initialize its DATA part to 9.

Take two pointer variables PTR and PREPTR and initialize them with START so that START, PTR, and PREPTR point to the first node of the list.

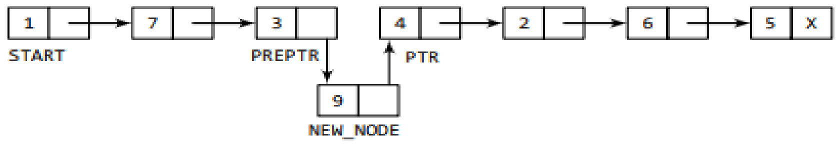


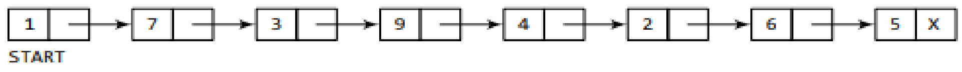
Move PTR and PREPTR until the DATA part of PREPTR = value of the node after which insertion has to be done. PREPTR will always point to the node just before PTR.





Add the new node in between the nodes pointed by PREPTR and PTR.

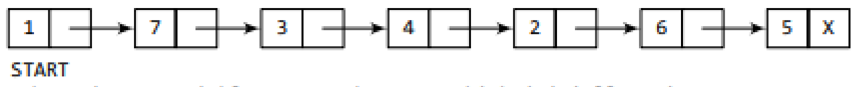




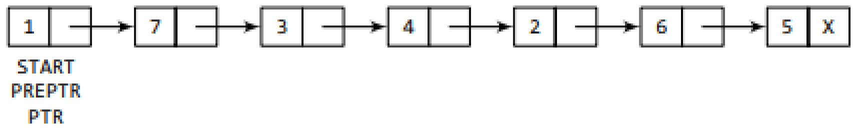
B) **Deleting a node at the end – 3 marks**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
       [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR->NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR->NEXT
       [END OF LOOP]
Step 6: SET PREPTR->NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

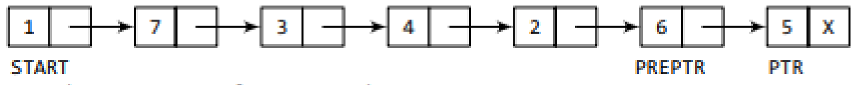
In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list. In the while loop, we take another pointer variable PREPTR such that it always points to one node before the PTR. Once we reach the last node and the second last node, we set the NEXT pointer of the second last node to NULL, so that it now becomes the (new) last node of the linked list. The memory of the previous last node is freed and returned back to the free pool.



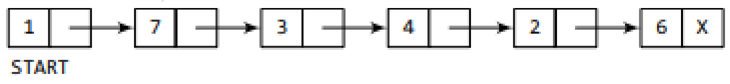
Take pointer variables PTR and PREPTR which initially point to START.



Move PTR and PREPTR such that NEXT part of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR.



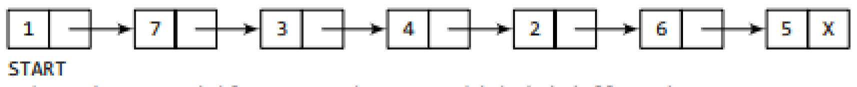
Set the NEXT part of PREPTR node to NULL.



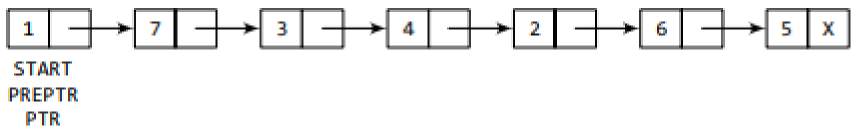
C) **Deleting a node before a given node. 3.5 marks**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
       [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR->DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR->NEXT
       [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR->NEXT = PTR->NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

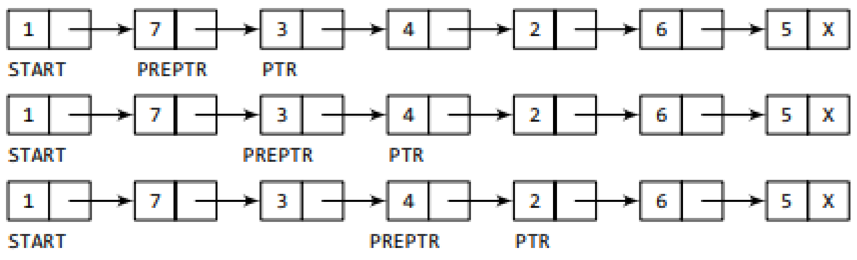
In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list. In the while loop, we take another pointer variable PREPTR such that it always points to one node before the PTR. Once we reach the node containing VAL and the node succeeding it, we set the next pointer of the node containing VAL to the address contained in next field of the node succeeding it. The memory of the node succeeding the given node is freed and returned back to the free pool.



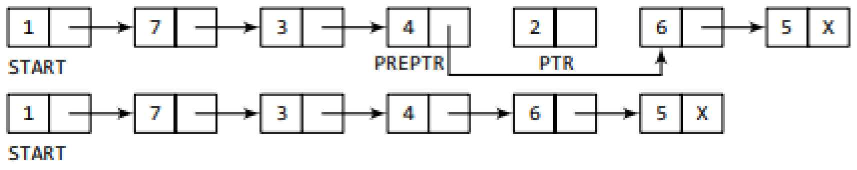
Take pointer variables PTR and PREPTR which initially point to START.



Move PREPTR and PTR such that PREPTR points to the node containing VAL
and PTR points to the succeeding node.



Set the NEXT part of PREPTR to the NEXT part of PTR.

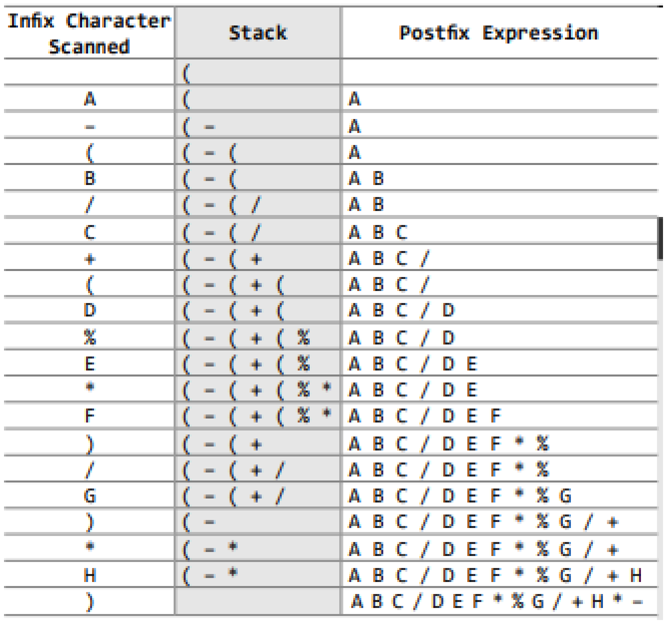


19a. Write the algorithm to convert infix expression to postfix expression and apply the same to convert the following infix expression: A – (B / C + (D % E * F) / G) * H).

Algorithm – 5 marks

Problem – 5 marks

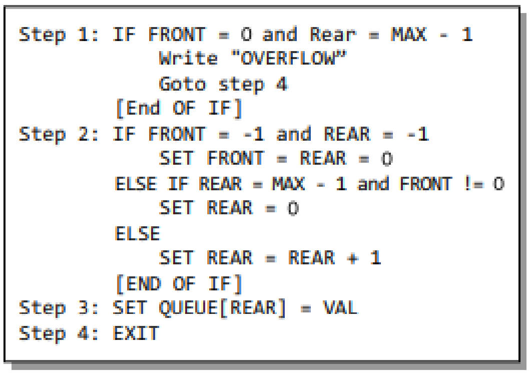
```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
       IF a "(" is encountered, push it on the stack
       IF an operand (whether a digit or a character) is encountered, add it to the
       postfix expression.
       IF a ")" is encountered, then
         a. Repeatedly pop from stack and add it to the postfix expression until a
            "(" is encountered.
         b. Discard the "(". That is, remove the "(" from stack and do not
            add it to the postfix expression
       IF an operator O is encountered, then
         a. Repeatedly pop from stack and add each operator (popped from the stack) to the
            postfix expression which has the same precedence or a higher precedence than O
         b. Push the operator O to the stack
       [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```



Infix Character Scanned	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
(	(- (	A
B	(- (	A B
/	(- (/	A B
C	(- (/	A B C
+	(- (+	A B C /
(	(- (+ (	A B C /
D	(- (+ (	A B C / D
%	(- (+ (%	A B C / D
E	(- (+ (%	A B C / D E
*	(- (+ (% *	A B C / D E
F	(- (+ (% *	A B C / D E F
)	(- (+	A B C / D E F * %
/	(- (+ /	A B C / D E F * %
G	(- (+ /	A B C / D E F * % G
)	(-	A B C / D E F * % G / +
*	(- *	A B C / D E F * % G / +
H	(- *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * -

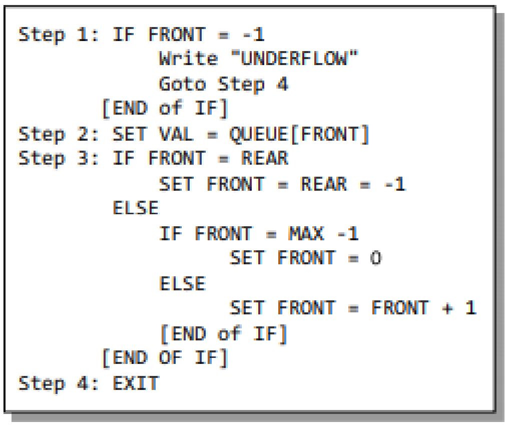
19b.Explain the insertion and deletion operation in circular queue with example and elaborate the two variants of Double ended queue.

Insertion in circular queue: 4 marks (with example)



```
Step 1: IF FRONT = 0 and Rear = MAX - 1
            Write "OVERFLOW"
            Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT != 0
            SET REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```

Deletion in circular queue: 4 marks (with example)



```
Step 1: IF FRONT = -1
            Write "UNDERFLOW"
            Goto Step 4
       [END of IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
            SET FRONT = REAR = -1
        ELSE
            IF FRONT = MAX -1
                  SET FRONT = 0
            ELSE
                  SET FRONT = FRONT + 1
            [END of IF]
       [END OF IF]
Step 4: EXIT
```

Two variants of double ended queue: 2 marks

Input restricted deque - In this dequeue, insertions can be done only at one of the ends, while deletions can be done from both ends.

Output restricted deque - In this dequeue, deletions can be done only at one of the ends, while insertions can be done on both ends.