ASSIGNMENT:-1

1 Design an algorithm to perform deletion operation by searching an item in an array A[..] having elements from lower bound LB to upper bound UB.

Ans [Consider an array A[..] having elements from lb to ub and item is given to be deleted from the array.]

Delete (A[..], lb, ub, item)

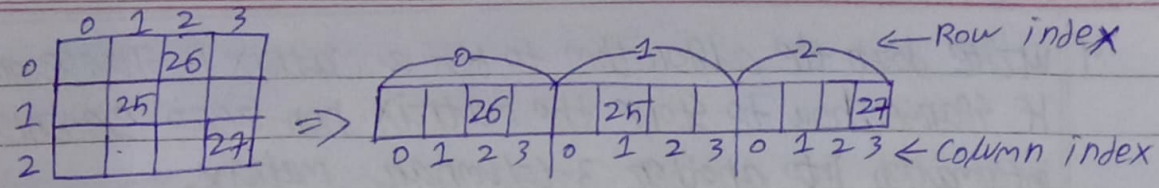
Step 1: Start
Step 2: Let i, flag=0
Step 3: for (i=lb to ub), incr by 1
  Step 3.1: If (A[i] = item), then
    Step 3.1.1: flag = 1
    Step 3.1.2: break
  [End of if]
[End of for]
Step 4: If (flag=0), then
  Step 4.1: Display "Item to be deleted is not found"
Step 5: else
  Step 5.1: while (i <= ub)
    Step 5.1.1: A[i] = A[i+1]
    Step 5.1.2: i = i+1
  [End of while]
  Step 5.2: ub = ub-1
[End of if]
Step 6: exit

2 write an algorithm to sort a list of elements present in an array X[0] in ascending order.

Ans [Consider an array X[0] having elements from lb to ub for sorting in ascending order.]
step 1: start
step 2: let i, j, temp
step 3: for (i=lb to ub-1), incr by 1
  step 3.1: for (j = i+1 to ub), incr by 1
    step 3.1.1: if (a[i] > a[j]), then
      step 3.1.1.1: temp=a[i], a[i]=a[j], a[j]=temp
    [end of if]
  [end of for]
[end of for]
step 4: exit

3 what is row-major order and column-major order? Given a matrix a[7][8] having base address 1000. If the size of each memory is 4 bytes, and we need to find the address of a[4][3] then find the address in row-major order and also in column-major order.

Ans ① Row-major order:- When the memory is occupied row by row serially then it is called the row major order. To find the address the formula is: A[i][j] = Base + w * (n*i + j)
where, Base = the starting or base address
w = the size of each element
n = the total number of columns
i and j = the corresponding row and column address we are finding.



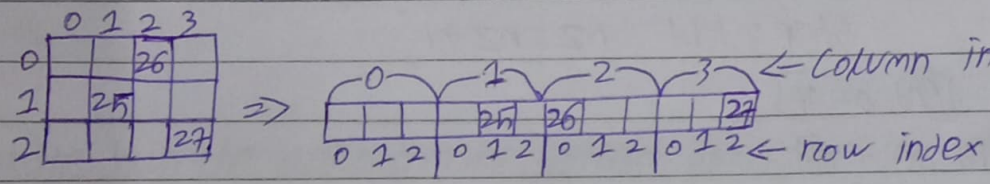
(II) Column major order :- when the memory of a matrix occupied serially column by column then it is called column major order. To find the address the formula is : $A[i][j] = Base + w * (i + m * j)$

where, Base = the starting or base address

w = the size of each element

m = the total number of rows

i and j = the corresponding row and column address we are finding.



Given, Base address = 1000, w = 4 bytes, m = 7, n = 8, i = 4, j = 3

The address in row-major order is :-

$a[i][j] = Base + w * (n * i + j)$

$a[4][3] = 1000 + 4 * (8 * 4 + 3)$

$= 1000 + 4 * 35$

$= 1000 + 140$

$= 1140$

The address in column-major order is :-

$a[i][j] = Base + w * (j + m * j)$

$a[4][3] = 1000 + 4 * (4 + 7 * 3)$

$= 1000 + 4 * 25$

$= 1000 + 100$

$= 1100$

4 Write down the algorithm to test a matrix is sparse or not. If sparse, how to store the matrix non-zero elements information into another 3-columnar matrix.

Ans [Consider a matrix A[m][n] having elements. Declare a 3-columnar matrix SP[nz+1][3] where nz is non-zeros count.]

Sparse (A[m][n], SP[nz+1][3])
Step 1: Start
Step 2: Let nz=0, i, j
Step 3: for (i=0 to m-1), incr by 1
  Step 3.1: for (j=0 to n-1), incr by 1
    Step 3.1.1: If (A[i][j] != 0), then
      Step 3.1.1.1: nz=nz+1
    [End of if]
  [End of for]
[End of for]
Step 4: If (nz >= (m*n)/2), then
  Step 4.1: Display "It is not sparse"
Step 5: else
  Step 5.1: Display "It is sparse"
  Step 5.2: Call Alternate (A[m][n], SP[nz+1][3])
[End of if]
Step 6: Stop

Alternate (A[m][n], SP[nz+1][3])
Step 1: Start
Step 2: Let i, j, k = 1
Step 3: SP[0][0] = m, SP[0][1] = n, SP[0][2] = nz
Step 4: for (i=0 to m-1), incr by 1
  Step 4.1: for (j=0 to n-1), incr by 1
    Step 4.1.1: if (A[i][j] ≠ 0), then
      Step 4.1.1.1: SP[k][0] = i, SP[k][1] = j, SP[k][2] = A[i][j]
      Step 4.1.1.2: k = k+1
    [end of if]
  [end of for]
[end of for]
Step n: exit

5 Write down the algorithm to perform PUSH and POP operations on a given stack using an array.

Ans Push :-
[Consider a stack[size], initially top = -1]
Push (stack[size], top, item)
Step 1: Start
Step 2: If (top = size-1), then
  Step 2.1: Display "Stack is full or overflow"
Step 3: else
  Step 3.1: top = top + 1
  Step 3.2: Stack[top] = item
[end of if]
Step 4: Stop

Pop :-
[Consider a stack[size] having elements and top contains index of top most element.]
Pop (stack[size], top)
step 1 : start
step 2 : if (top = -1), then
  step 2.1 : Display "stack is empty on underflow"
step 3 : else
  step 3.1 : let temp = stack[top]
  step 3.2 : Display "popped value is", temp
  step 3.3 : top = top - 1
[end of if]
step 4 : stop

6 write down the algorithm to convert a given infix expression into equivalent postfix notation using stack.

Ans [Given an infix expression Q and we need to find equivalent postfix expression P. Declare a stack where top represents top most element. Here i is the index for Q and j is the index for P.]
InfixtoPostfix (P, j, Q, i, stack, top)
step 1 : start
step 2 : push '(' to stack and put ')' at the end of Q
step 3 : while (stack is not empty)
  step 3.1 : Scan symbol for Q[i]
  step 3.2 : if Q[i] is operand then put the operand at the end of P[j]
  step 3.3 : else if Q[i] is '(' then push '(' into the stack [top]
  step 3.4 : else if Q[i] is operator ⊗, then

Step 3.4.1 : while (stack[top] is operator AND
Priority (stack[top]) >= Priority (Q[i]))
Step 3.4.1.1 : Pop operator from stack[top] and
put at the end of P[j].
[End of while]
Step 3.4.2 : Push operator Q[i] into stack[top]
Step 3.5 : else if Q[i] is ')' then
Step 3.5.1 : while (stack[top] not equal to '(')
Step 3.5.1.1 : Pop operator from stack[top] and
put at the end of P[j]
[End of while]
Step 3.5.2 : Delete '(' from stack[top]
[End of if]
[End of while]
Step 4 : Display "Postfix Expression is", P
Step 5 : Stop

7 write down the algorithm to perform Insertion and deletion operations on a linear queue implemented using an array.

Ans Insertion Algorithm :-

[Consider a queue[size], Front = -1, Rear = -1 initially. Given an item for insertion]
Insertion (Q[size], Front, Rear, Item)
Step 1 : Start
Step 2 : if (Rear = Size-1), then
Step 2.1 : Display "queue overflow"
Step 3 : else if (Front = -1 AND Rear = -1), then
Step 3.1 : Front = 0, Rear = 0
Step 3.2 : Q[Rear] = Item
Step 4 : else
Step 4.1 : Q[++Rear] = Item
[End of if]
Step 5 : Stop

Deletion Algorithms:-

[Consider a queue Q[size], initially Front = -1, Rear = -1. Temp declared to hold the deleted value.]

Deletion (Q[size], Front, Rear, Temp)

Step 1: Start

Step 2: if (Front = -1 AND Rear = -1), then

    step 2.1: Display "queue underflow"

Step 3: else if (Front = Rear), then

    step 3.1: Temp = Q[Front]

    Step 3.2: Front = -1, Rear = -1

    Step 3.3: Display "Deleted value is", temp

Step 4: else

    Step 4.1: Temp = Q[Front]

    Step 4.2: Front = Front + 1

    Step 4.3: Display "Deleted value is", temp

[End of if]

Step 5: Stop

8 given infix expression Q = A-S/D+(E*F^G)-H
Find its equivalent postfix using stack.

Ans First put ')' at the end of infix expression Q and push '(' to stack[Top].

Symbol	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
S	(-	AS
/	(-/	AS
D	(-/	ASD
+	(+	ASD/-

(	(+(	ASD/-
E	(+(	ASD/-E
*	(+(*	ASD/-E
F	(+(*	ASD/-EF
^	(+(*^	ASD/-EF
G	(+(*^	ASD/-EFG
)	(+	ASD/-EFG^*
-	(-	ASD/-EFG^*+
H	(-	ASD/-EFG^*+H
)	empty	ASD/-EFG^*+H-

9 Given two arrays A[5] and B[7] having elements in ascending order. Design an algorithm to apply merging operation on both arrays and generate the resultant list into array C[12] in ascending order.

Ans [Consider A[5] and B[7] having elements in ascending order from lb1 to ub1 and lb2 to ub2 and a resultant array C[12] having lb3 and ub3]

Merging (A[5], lb1, ub1, B[7], lb2, ub2, C[12], lb3, ub3)

Step 1: Start

Step 2: Let i, j, k

Step 3: i=lb1, j=lb2, k=0

Step 4: while (i<=ub1 AND j<=ub2)

  Step 4.1: if (A[i] < B[j]), then

    Step 4.1.1: C[k] = A[i], i=i+1, k=k+1

  Step 4.2: else

    Step 4.2.1: C[k] = B[j], j=j+1, k=k+1

  [end of if]

[end of while]

Step 5: while (j <= ub1)

  Step 5.1: C[k] = A[i], k=k+1, i=i+1

[end of while]

Step 6: while (j <= ub2)
Step 6.1: C[k] = B[j], k=k+1, j=j+1
[end of while]
Step 7: lb3 = 0, ub3 = k-1
Step 8: Stop

10 write down the algorithm to evaluate a given ~~postfix~~ expression using stack.

Ans Consider a postfix expression P given for evaluation.
Declare stack[size], top, i, A, B, R.
PostfixEvaluation (P, stack, i, top, A, B, R)
Step 1: Start
Step 2: Put ')' at the end of postfix P.
Step 3: while (P[i] != ')')
Step 3.1: If (P[i] is operand), then
Step 3.1.1: push P[i] into stack[top]
Step 3.2: else if (P[i] is operator) then
Step 3.2.1: A = Pop 1st element from stack
Step 3.2.2: B = Pop 2nd element from stack
Step 3.2.3: R = B P[i] A
Step 3.2.4: push R into stack[top]
[end of if]
[end of while]
Step 4: Display "Result is", stack[top]
Step 5: Stop

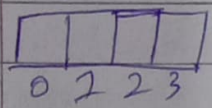
11 Given a stack STK[size] where size=4. Initially the top=-1. Apply the below list of operations on the stack and elaborate the execution process in detail. Push(10), Push(20), POP(), Push(30), Push(40), Push(50), Push(60), POP(), POP(), POP(), Push(70).

Ans

Initially:-

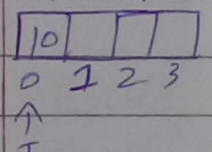
STK[4]

TOP=-1



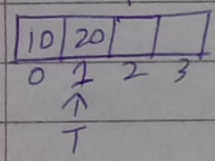
Push(10):-

TOP=~~-1~~ 0



Push(20):-

TOP=0 1

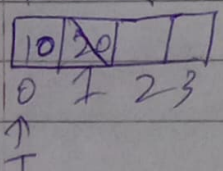


POP():-

TOP=1

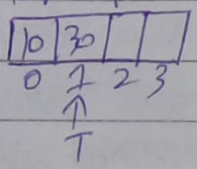
let temp=STK[TOP]
=20

TOP=~~1~~ 0



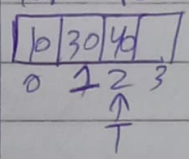
Push(30):-

TOP=0 1



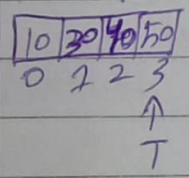
Push(40):-

TOP=~~1~~ 2



Push(50):-

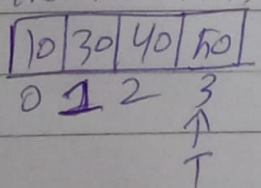
TOP=~~2~~ 3



Push(60):-

TOP=3=size-1

As TOP is equal to size-1. Therefore the stack is full.

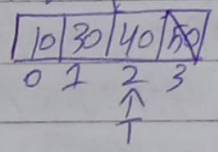


POP():-

TOP=3

let temp=STK[TOP]=50

TOP=~~3~~ 2

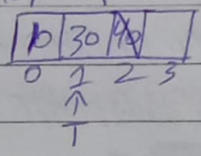


POP():-

TOP=2

let temp=STK[TOP]=40

TOP=~~2~~ 1

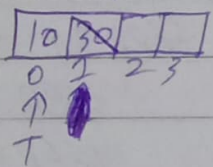


POP():-

TOP=1

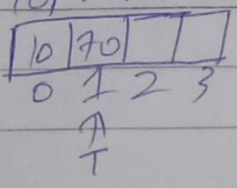
let temp=STK[TOP]=30

TOP=~~1~~ 0



Push(70):-

TOP=0 1



12 Given a list of elements from lb to ub in an array A[100]. write two algorithms to perform:
(a) Insertion of an item at a specific location
(b) Search for a given item.

Ans (a) Given an array A[100] having elements from lb to ub and an item to be inserted at a specific location between lb to ub]

Insertion (A[100], lb, ub, item, position)
step 1: start
step 2: let i
step 3: for (i = ub to position), decr by 1
  step 3.1: A[i+1] = A[i]
[end of for]
step 4: A[position] = item
step 5: ub = ub+1
step 6: stop

(b) [Given an item to search for its position. Here let us search whether the element is exist or not.]

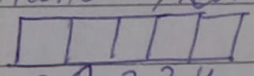
search (A[100], lb, ub, item)
step 1: start
step 2: let i, flag=0
step 3: for (i=lb to ub), incr by 1
  step 3.1: if (A[i] = item), then
    step 3.1.1: flag = 1
    step 3.1.2: break
  [end of if]
[end of for]

Step 4: if (flag=0), then
Step 4.1: Display "Item is not found"
Step 5: else
Step 5.1: Display "Item found"
[End of IF]
Step 6: Stop

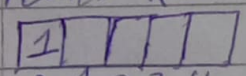
13 Given a linear queue QUEUE[size] where size=5.
Initially Front=-1 and Rear=-1
Apply the below lits of operations on the queue and elaborate the execution process in details. Insert(1), Insert(2), Insert(3), Delete(), Delete(), Insert(4), Insert(5), Delete(), Insert(6), Insert(7).

Ans

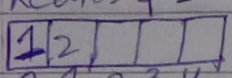
Initially:-
Queue[5]
Front=-1, Rear=-1



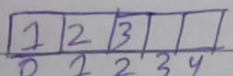
Insert(1):-
Front=~~-1~~ 0
Rear=~~-1~~ 0



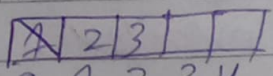
Insert(2):-
Front=0
Rear=~~0~~ 1



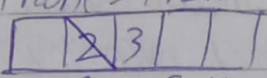
Insert(3):-
Front=0, Rear=~~1~~ 2



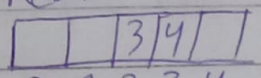
Delete():-
~~Delete~~
Front=0
Rear=2
Let temp=QUEUE[Front]
=1
Front=Front+1=1



Delete():-
Front=1
Rear=2
Let temp=QUEUE[Front]
=2
Front=Front+1=2



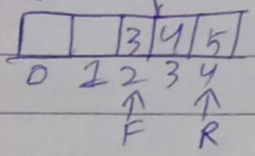
Insert(4):-
Front=2
Rear=~~2~~ 3



Insert(5):-

Front = 2

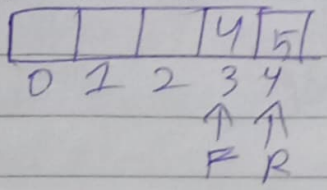
Rear = 3 4



Insert(6):-

Front = 3, Rear = 4

As Rear = size - 1. Therefore stack is full on overflow



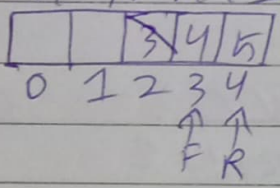
Delete():-

Front = 2

Rear = 4

let temp = QUEUE[Front]
= 3

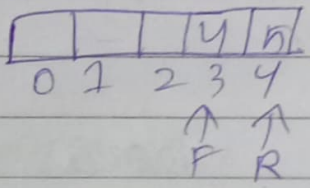
Front = Front + 1 = 3



Insert(7):-

Front = 3, Rear = 4

As Rear = size - 1. Therefore stack is full on overflow



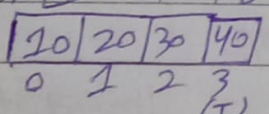
14 List out:-

ⓐ The stack overflow and underflow conditions and

ⓑ Linear queue overflow and underflow conditions.

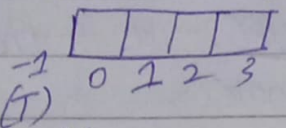
Ans ⓐ Stack overflow:- When the stack is full then "top = SIZE-1", then it is called stack overflow.

Let Stack[4] having elements.



If we try to insert (80), then it will not be possible and it will display "Stack overflow".

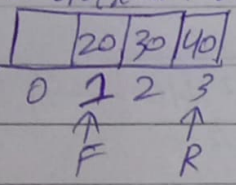
Stack underflow :- when the "top = -1 or NULL", then the stack is empty or underflow. Let stack[4] is declared with the top = -1 initially



If we try to delete(), then it will not be possible and it will display "stack underflow".

(b) Linear queue overflow :

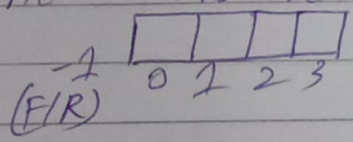
when the queue is full then "Rear = size-1", then it is called queue overflow. Let a queue[4] having elements



If we try to insert(50), then it will not be inserted and it will display "queue overflow".

Linear queue ~~over~~ underflow :-

when the "Front = -1 AND Rear = -1", then the queue is empty or underflow. Let a queue[4] is declared with the Front = -1 and Rear = -1 initially



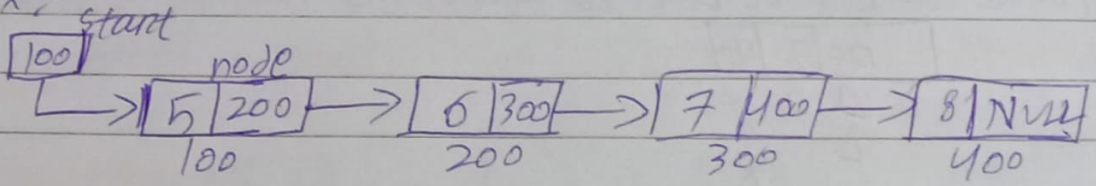
If we try to delete(), then it will not be executed and it will display "queue underflow".

15 Define the terms:
● Abstract Datatype, Linked list, Queue, Tree, Stack.

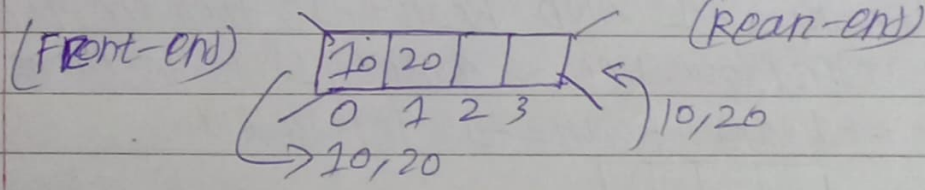
Ans Abstract Datatype:- It is the most suitable datatype for a data model.

Linked List:- It is the collection of non-consecutive nodes connected in serial manner. A node contains 2 parts:-
ⓐ Info part:- It hold the value
ⓑ Linked part:- It hold the address of next node.

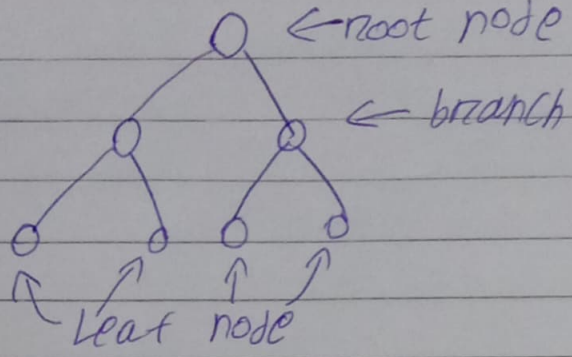
EX:-



Queue:- A Linear datastructure (i.e. either array or linkedlist) which implements "First in First out" concept (FIFO). Let us consider a Q[4] having elements.



Tree:- A group of memory locations (node) in which if there is hierarchical relationship exists or top-to-down relationship exists among them, then it is called Tree. Here the top most memory is the root node. From root node it is connected to different nodes called branches. At the bottom level whatever the nodes available called leafs (leaf node). Let us consider a binary tree.



Stack :- A linear datastructure that impliments "Last In First out" concept is called stack. The concept of stack can be implemented using either array or linked list. Let us consider S[4] having elements