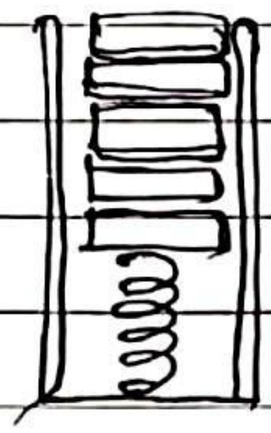
Stacks and Queues

- **Introduction to stacks :**



→ push the plate from the top (insert) and take the plate from the top (pop).

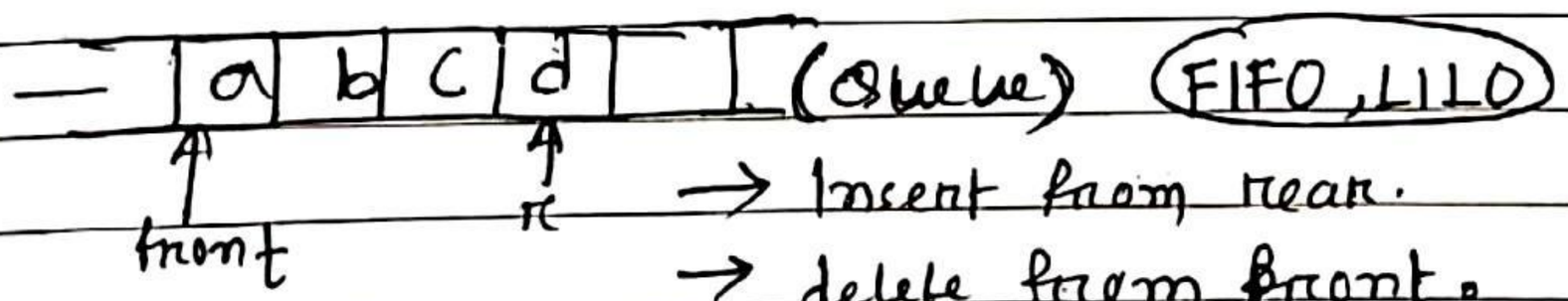
(stack) (FILO, LIFO)

- **Applications of stack :**

(i) Recursion.
(ii) IF → PF conversion.
(iii) Parsing
(iv) Browsers. →
(v) Editors. (use)
(vi) Tree traversals and graph traversal.

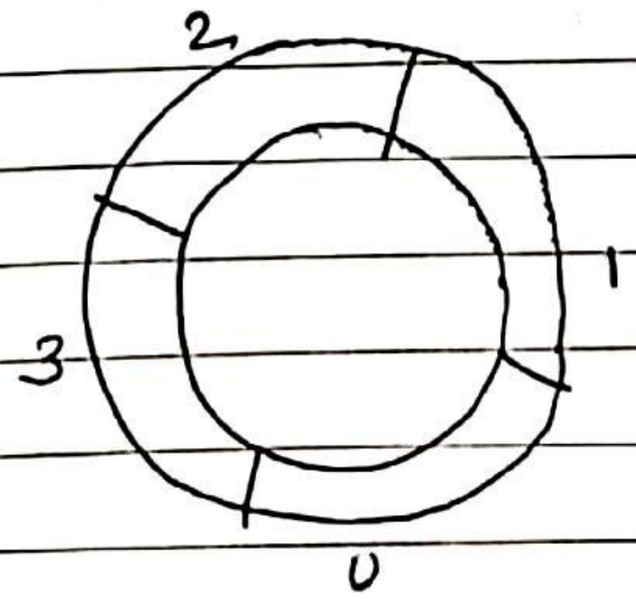
- **Implementation of a Queue using circular array =**

→ Insert the element from one side (enqueue) and delete the element from other side (dequeue).



(Queue) (FIFO, LILO)

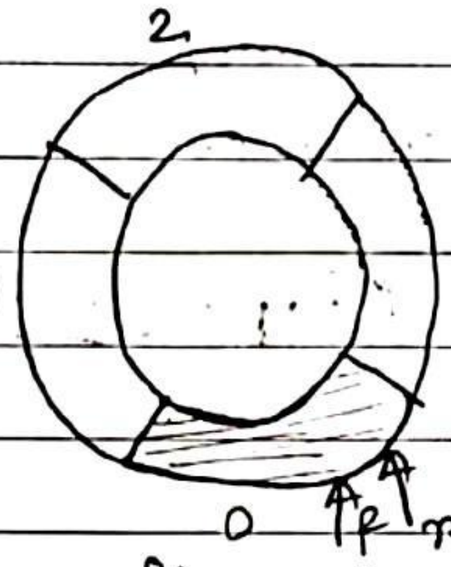
→ Insert from rear.
→ delete from front.



circular queue.

if → In circular queue, if the size of an array 'n' then we use '(n-1)' amount of size.

~~→ one less~~



circular queue.

→ should a blank space be

① Inserting an item in queue =

```
enqueue(item)
{
  rear = (rear+1) mod n;

  if(front == rear)
   { pf("Q is full");

    if(rear==0) rear=n-1;
    else
  Else rear = rear-1;
   return;
  }

  else {
       q[rear] = item;
       return;
  }
```

② delete an item from Queue :-

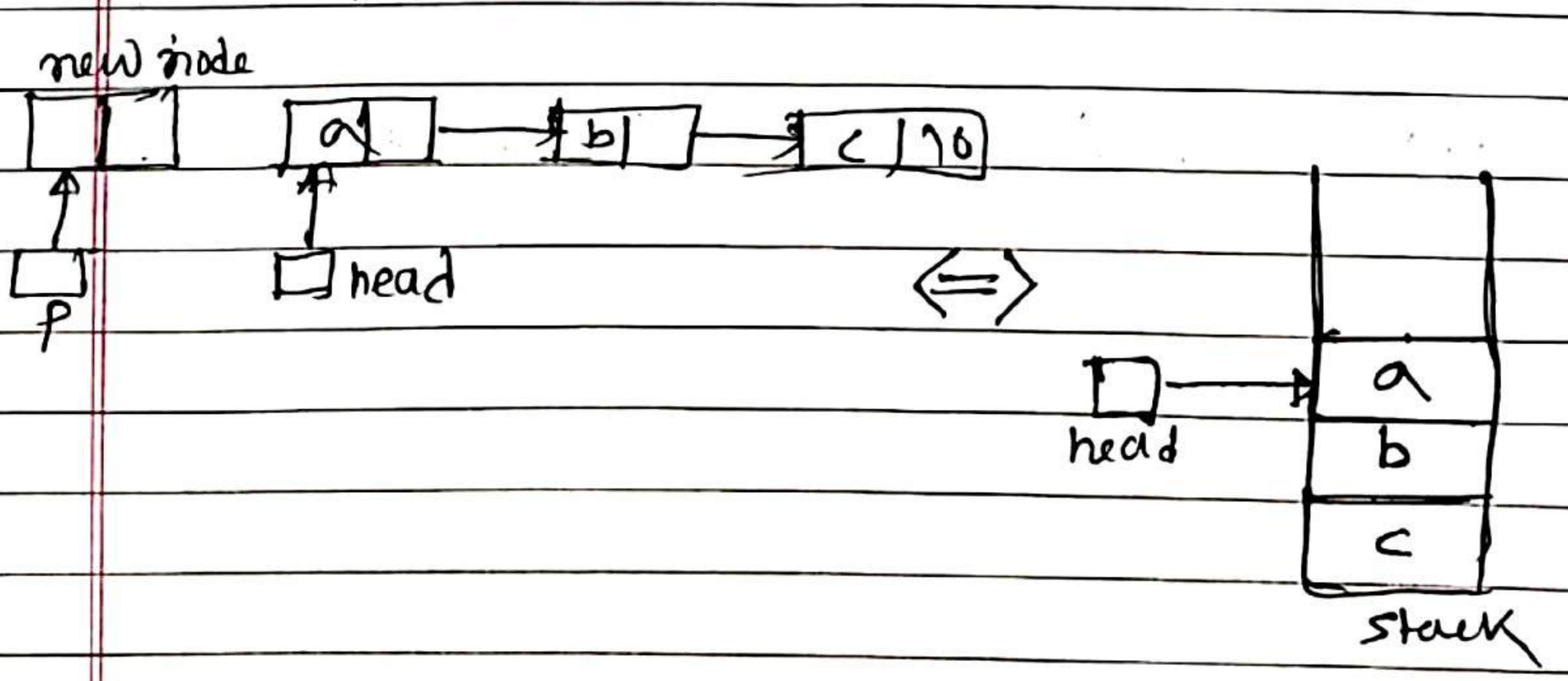
```
int Dequeue()
{
    if (front == rear)
    {
        Pf ("Q is empty");
        return -1;
    }
    else
    {
        front = (front+1)%n;
        item = q[front];
        return item;
    }
}
```

Over flow :
(rear+1) mod n == front

Under flow
front == rear

- Linked list Implementation of stack :-

```
struct node
{
    int i;
    struct node *link;
};
```



(1) Inserting a new item =

```
push (int item)
{
  struct node *p = (struct node*) malloc (sizeof(struct node));
  if (p == NULL) { pf("error of malloc"); return;}

  p → data = item;
  p → link = ~~head~~ NULL;
  p → link = head;
  head = p;
}
```

Time complexity = O(1) (constant time)

(2) deletion a item from stack =

~~Pseudocode~~

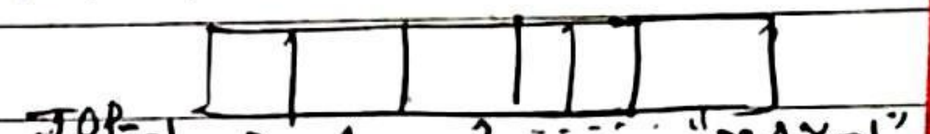
```
int pop()
{ struct node *p;
  int item;
  if(head == NULL) { pf("Underflow"); return -1;}

  item = head → i;
  p = head;
  head = head → next;
  free (p);
```

Time complexity = O(1) (constant time)..

• Implementation of stack using array =

```
int stack[MAX];
int top = -1; //(stack Empty)
```



• Puring a item =

```
void push (int item)
{
    if (top == MAX-1)
        pf("overflow");

    else {
        top = top+1;
        stack[top] = item;
    }
    return;
}
```

stack[++top]= item;

• poping a item =

```
int pop()
{ int temp;
  if (top == -1)
  { pf("underflow");
    return -1;
  }
else
{
    temp = stack[top];
    top = top-1;
}
return temp;
}
```

temp = stack[top--];

→ Time complexity (push, pop) = O(1) Time.

classmate
Date
Page

cr-2012

Question -1

Suppose a circular queue of capacity (n-1) elements is implemented with an array of 'n' elements. Assume that the insertion and deletion operations are carried out using 'REAR' and 'FRONT' as array index variables respectively, Initially, REAR=FRONT=0. The Conditions to detect queue ~~with~~ full and queue empty are =

→ Queue Full:

(REAR+1) mod n == FRONT.

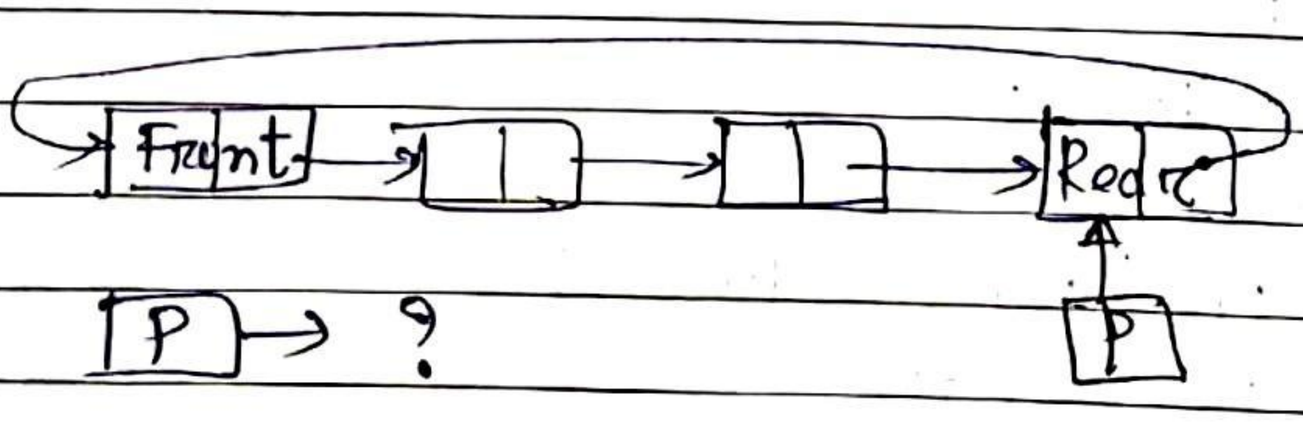
Queue Empty:

REAR = FRONT

cr-2004

Question - 2

A circularly linked list is used to represent a Queue. A single variable 'p' is used to access the Queue. To which node should 'p' point to such that both the operations enQueue and deQueue can be performed in constant time?



a) Read node
b) FRONT node
c) not possible.
d) node next to Front.

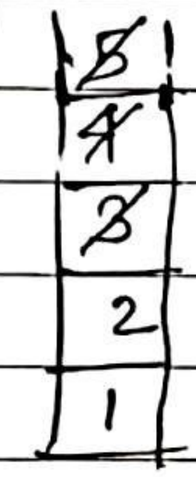
→

Question-3

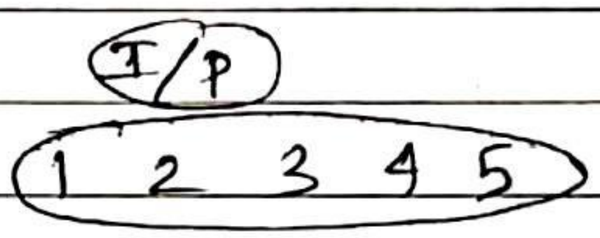
Which of the following permutations can be obtained in the o/p (in the same order) using a stack assuming that the i/p is the sequence. 1, 2, 3, 4, 5 in the order?

✗ a) 3, 4, 5, 1, 2. ✓ b) 3, 4, 5, 2, 1.
✗ c) 1, 5, 2, 3, 4. ✗ d) 5, 4, 3, 2, 1, 2.

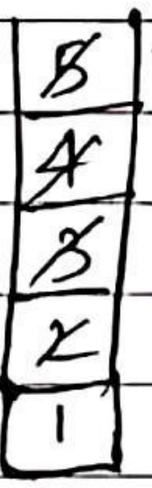
→ ✗ (a)



O/P
3, 4, 5, . .



✓ (b)



O/P
3, 4, 5, 2, 1

✗ (c)

1, 5, . , . .

✗ (d)

5, 4, 3,

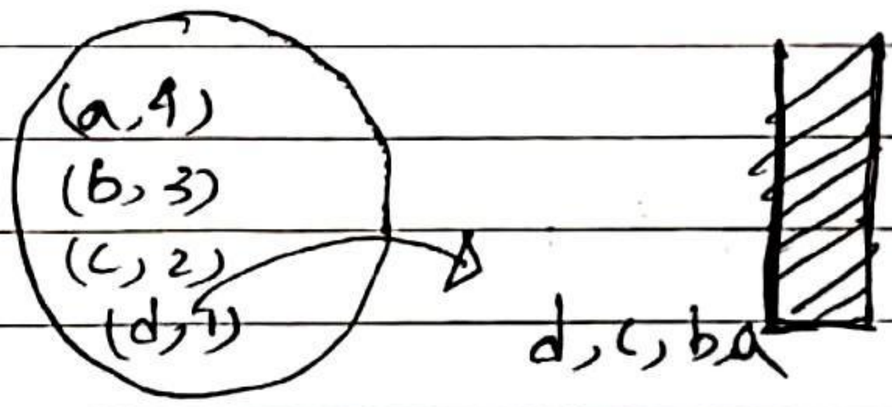
G-1997

Question - 4

A priority queue 'Q' is used to implement a stack 's' that stores characters PUSH(c) is implemented as ~~Insert~~ Insert (Q, c, K) where K is an appropriate integer key chosen by the implementation. POP is implemented as DELETE MIN(Q). For a sequence of PUSH operations, the keys chosen are in.

a) Non - decreasing order.
b) Non - increasing order.
c) strictly increasing order.
d) strictly decreasing order.

→

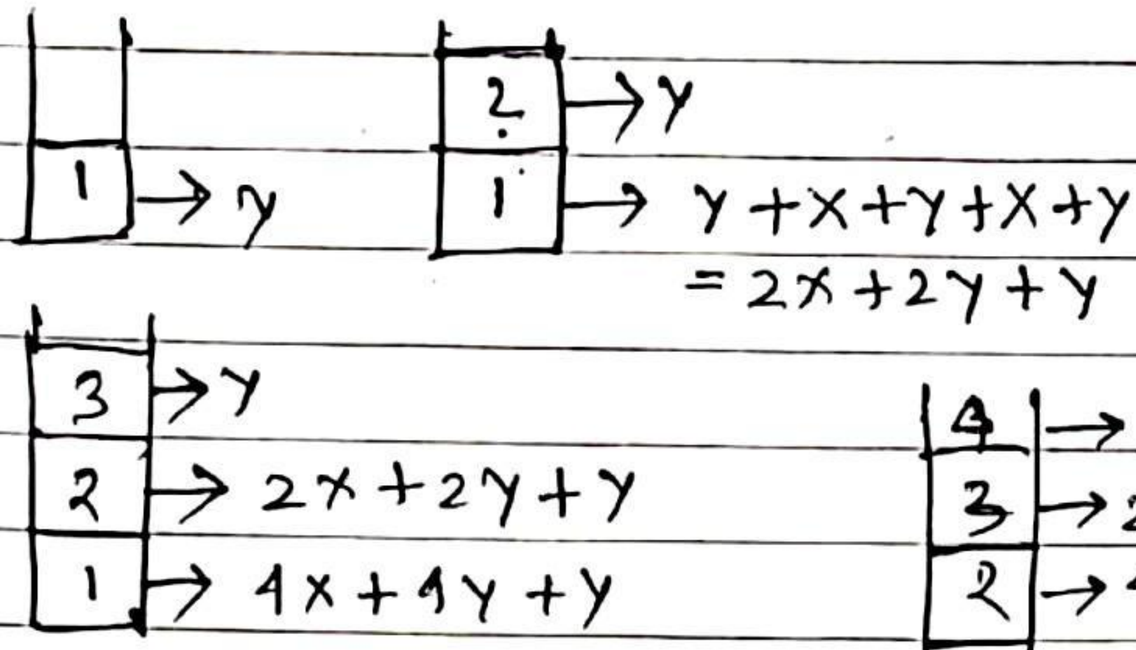


G-2003

Question - 5

Let 's' be a stack of size $n \geq 1$. starting with the ~~ent~~ empty stack, suppose we push the first 'n' natural numbers in sequence and then perform 'n' pop operations. Assume that push and pop operations take 'x' seconds each, and 'y' seconds elapse between the end of one such stack operation and the start of the next operation. For $m \geq 1$, define the stack-life of 'm' as the elapsed from end of push(m) to the start of the pop operation that removes 'm' from s. The average stack-life of an element of this stack is:

→



for n elements,

average life time,

$$\frac{2*\left(\sum_{i=1}^{n-1} i\right)x + 2\left(\sum_{i=1}^{n-1}\right)y + ny}{n}$$

$$\Rightarrow \frac{2*\frac{(n-1)(n-1+1)}{2}*x + 2*\frac{(n-1)(n-1+1)}{2}*y + ny}{n}$$

$$\Rightarrow \frac{n(n-1)x + n(n-1)y + ny}{n}$$

$$\Rightarrow (n-1)x + (n-1)y + y$$

$$\Rightarrow (n-1)(x+y) + y$$

$$\Rightarrow n(x+y) - x - y + y$$

$$\Rightarrow \boxed{n(x+y) - x}$$

06-2006

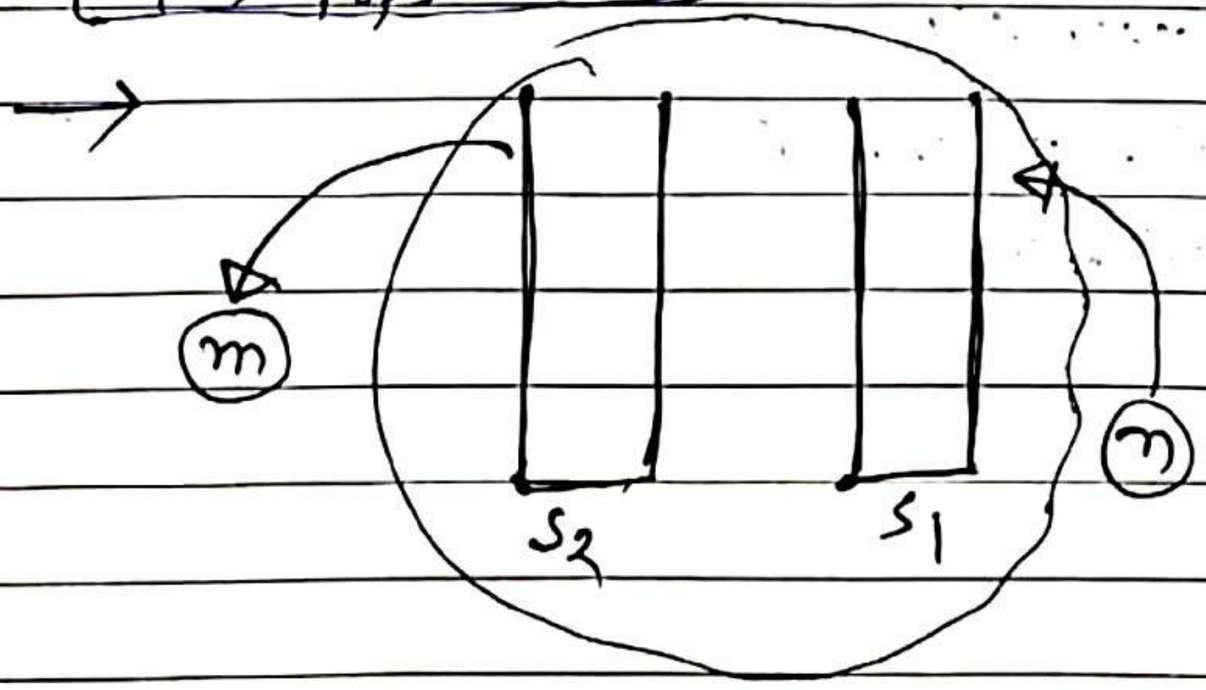
Question-6

Queue is implemented using two stacks S_1 and S_2 as given below:

```
void insert(Q, x) {
    push(S1, x);
    }
void delete(Q) {
    if (stack-empty (S2)) then
    if (stack-empty (S1)) then {
     printf ("Q is empty");
     return;
    }
  else while (!(stack-empty (S1)) {
        x = pop(S1);
        PUSH(S2, x);
    }
    x = pop(S2);
}
```

n → Inserts
m (≤n) → deletes.
x → Pushes
y → pops

what is relation b/w all this?



In Best case :

PUSH : $2m + (n-m)$
$= m+n$

POP : $2m$

In Worst case :

PUSH : $2n$

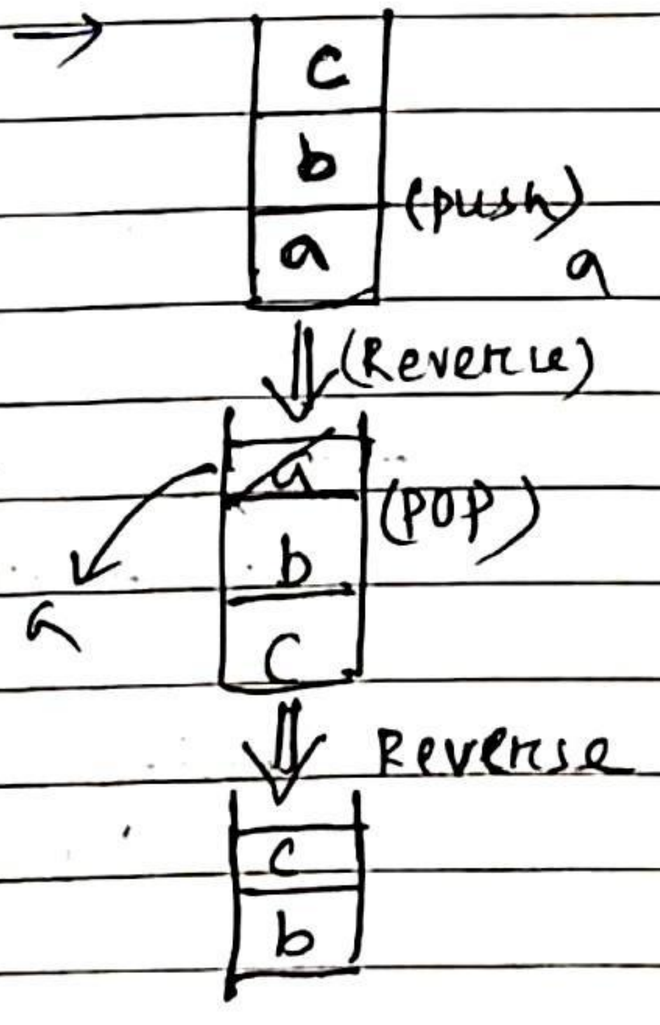
POP : $2(n+m)$

$$m+n \leq x \leq 2n$$
$$2m \leq y \leq n+m$$

gate-2000 **Question - 7**

Suppose a stack implementation supports, in addition to PUSH, and POP, an operation "REVERSE" the order of the elements on the stack.

Implement a queue using the above stack implementation, show how to implement "ENQUEUE" using a single operation and "DEQUEUE" using a sequence of 3 operations.



Enqueue → push

Dequeue → Reverse → POP → Reverse

Infix → postfix

- Infix to postfix Conversion Algorithm :- (Operator stack)

→ postfix related anything stack is required.

example:

① a+b → Infix expression:
ab+ → postfix expression.

② a+b*c → Infix exp
= a+bc*
= abc*+ → postfix expression.

③ a+b-c
= ab+-c
= (ab+c-)

④ a+(b-c) - Infix
= a+bc-
= abc-+ — postfix.

⑤ a+b*c - infix
= a+bc*
= (abc*+) — postfix

⑥ a + b * c - Infix.
↑ ↑ ↑ ↑ ↑

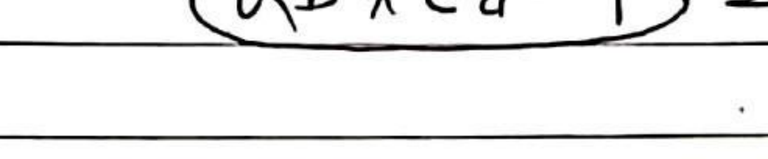
Stack: [+] [*]

= (abc*+) - Postfix.

+ <. (
(<. +

→ you can push an operator, only when the inside one is having lesser precidence.

⑦ a * b + c — Infix expression.



a b * c + — postfix expression.

⑧ a * b + (c − d) — Infix

a b * c d − + — postfix expression.

Algorithm

a) Creat a stack.
b) For each character 't' in the i/p.

```
{
  if ('t' is an operand)
      output 't';

  else if ('t' is a right parenthesis)
  {
      Pop and output tokens until a left parenthe-
      -sis is popped (but don't o/p)

  else // 't' is an operator or left parenthesis
  {
      pop and o/p tokens until one of lower
      priority than 't' is encounted or a
      left paren-thesis is encounted or
      the stack is empty.
  }
  push t
}
```

c) pop and o/p all the tokens untill the stack is empty.

space
or
Time complexity — O(n)
Data structure used — stack
~~or~~ PUSH only — Operators.

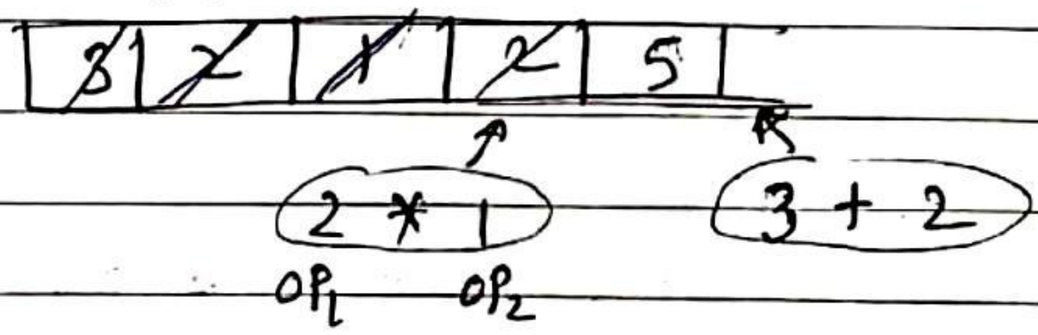
Postfix → Infix

- Postfix evaluation algorithm = Operand stack

~~Eg~~ 3+2*1 — Infix

3+21*

321*+ — postfix



PUSH — only operand.

Algorithm for postfix evaluation =

1. Scan the postfix string from left to right.
2. Initialize an empty stack.
3. Repeat steps 4 and 5 till all the characters are scanned.
4. If the scanned character is an operand, push it onto the stack.
5. If the scanned character is an operator, and if the operator is unary, then pop an element

from the stack. If the operator is binary, then pop two elements from the stack. After popping the elements ~~from the stack~~ apply the operator to those popped elements. push the result on to the stack.

6) After all the elements are scanned, the result will be in the stack.