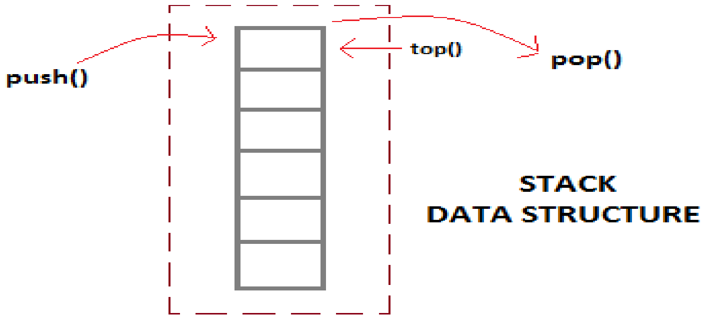
Sub:- 301 (Data Structure and Algorithm)

Second Class Test

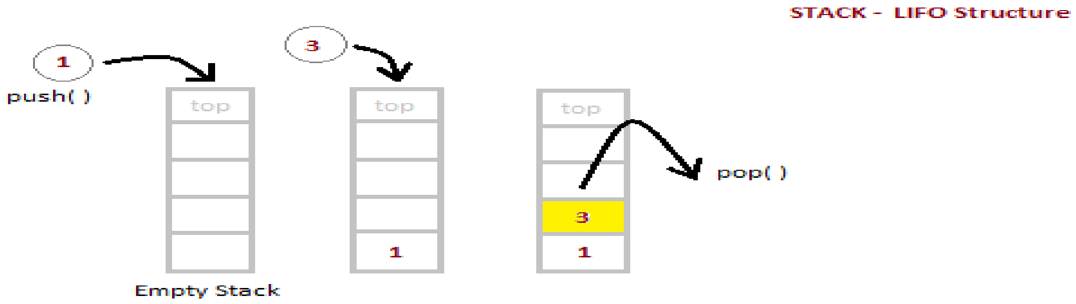
Q1:- What is Stack? Explain memory representation of stack and also explain PUSH, POP algorithm.

Ans:- Stack is an abstract data type with a bounded (predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack; the only element that can be removed is the element that was at the top of the stack, just like a pile of objects.

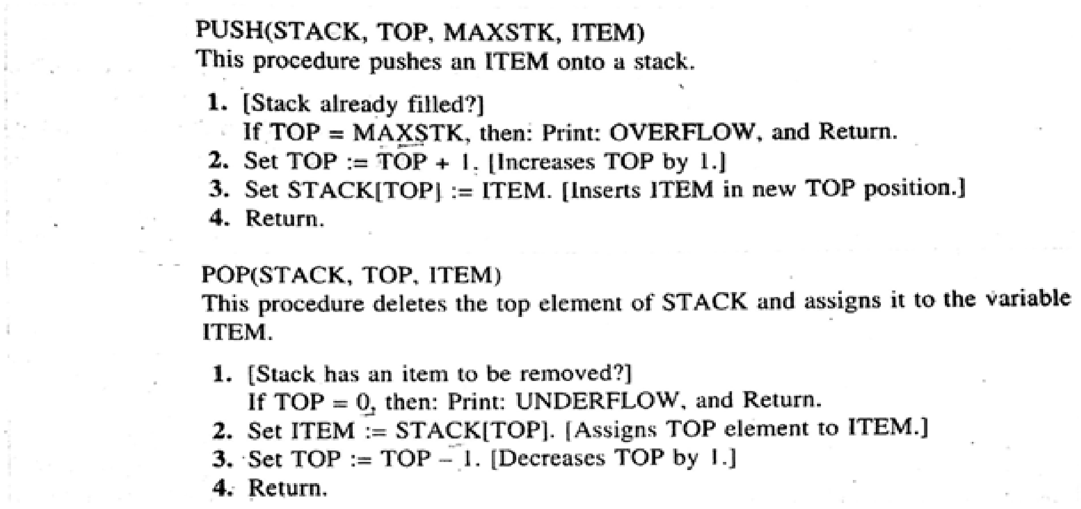


Implementation of Stack

Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size. Here we will implement Stack using array.



In a Stack, all operations take place at the "**top**" of the stack. The "**push**" operation adds an item to the top of the Stack.
The "**pop**" operation removes the item on top of the stack.



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.
```

Q2.Evaluate the following expression into Postfix Expression.

Q=A+(B*C-(D/E^F)*G)H

Sol:

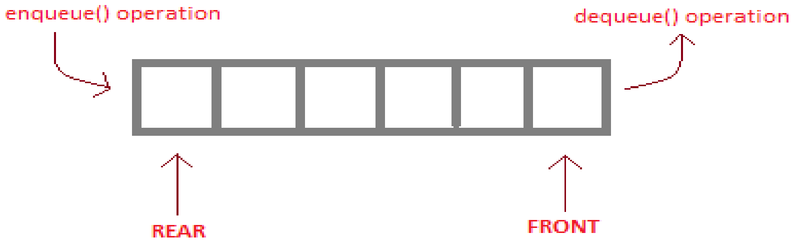
Step	**Symbol Scanned**	**Stack**	**Postfix Expression**
1.	A	_	A
2.	+	+	A
3.	(	+(	A
4.	B	+(	AB
5.	*	+(*	AB
6.	C	+(*	ABC
7.	-	+(-	ABC*
8.	(	+(-(	ABC*
9.	D	+(-(	ABC*D
10.	/	+(-(/	ABC*D
11.	E	+(-(/	ABC*DE
12.	^	+(-(/^	ABC*DE
13.	F	+(-(/^	ABC*DEF
14.	)	+(-(/^)	ABC*DEF^/
15.	*	+(-*	ABC*DEF^/
16.	G	+(-*	ABC*DEF^/G
17.	)	+(-*)	ABC*DEF^/G*-
18.	*	+*	ABC*DEF^/G*-
19.	H	+*	ABC*DEF^/G*-H
20.	_	_	ABC*DEF^/G*-H*+

Postfix Expression:ABC*DEF^/G*-H*+

Q3.What is Queue? Explain Memory representation of queue and also explain FRONT and REAR algorithm.

Ans:- Queue is also an abstract data type or a linear data structure, in which the first element is inserted from one end called **REAR**(also called tail), and the deletion of existing element takes place from the other end called as **FRONT**(also called head). This makes queue as FIFO(First in First Out) data structure, which means that element inserted first will also be removed first.

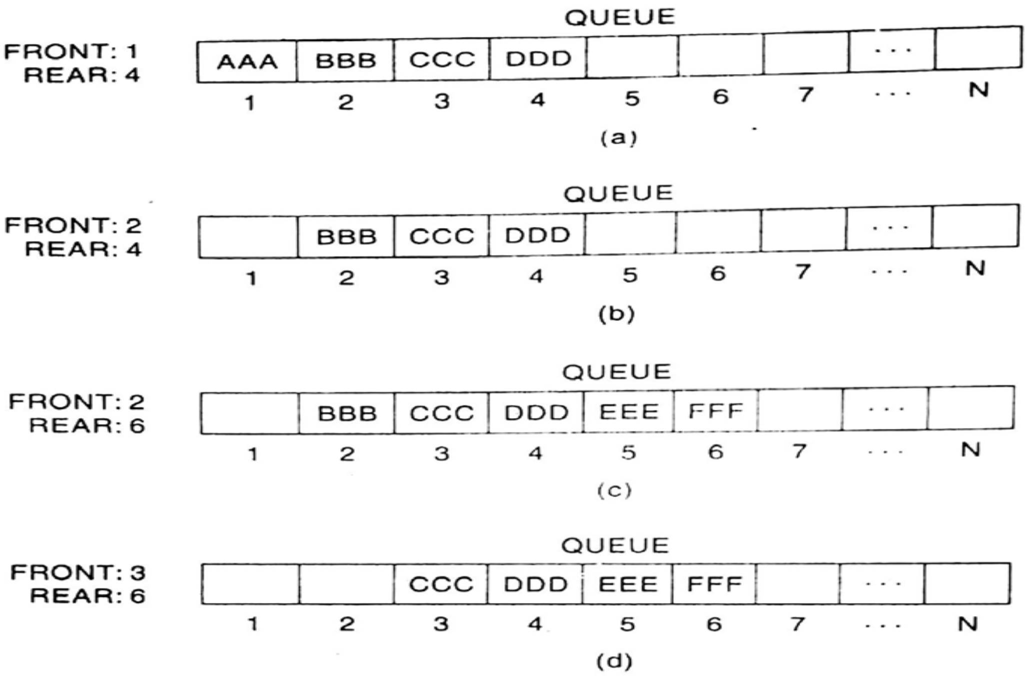
The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE



Array Representation of a Queue

QINSERT(QUEUE, N, FRONT, REAR, ITEM)

This procedure inserts an element ITEM into a queue.

```
1. [Queue already filled?]
   If FRONT = 1 and REAR = N, or if FRONT = REAR + 1, then:
       Write: OVERFLOW, and Return.
2. [Find new value of REAR.]
   If FRONT := NULL, then: [Queue initially empty.]
       Set FRONT := 1 and REAR := 1.
   Else if REAR = N, then:
       Set REAR := 1.
   Else:
       Set REAR := REAR + 1.
   [End of If structure.]
3. Set QUEUE[REAR] := ITEM. [This inserts new element.]
4. Return.
```

QDELETE(QUEUE, N, FRONT, REAR, ITEM)

This procedure deletes an element from a queue and assigns it to the variable ITEM.

```
1. [Queue already empty?]
   If FRONT := NULL, then: Write: UNDERFLOW, and Return.
2. Set ITEM := QUEUE[FRONT].
3. [Find new value of FRONT.]
     If FRONT = REAR, then: [Queue has only one element to start.]
         Set FRONT := NULL and REAR := NULL.
     Else if FRONT = N, then:
         Set FRONT := 1.
     Else:
         Set FRONT := FRONT + 1.
   [End of If structure.]
4 Return.
```