- Assignment Name: Lab4
- Description: Question based on queues and stacks
- Total Marks: 10
- Deadline: Depends on respective lab hrs

- Problem Title: **Implementing queue using two stacks**
- Marks allotted: 10
- Description:

A queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.

We define a queue to be a list in which all additions to the list are made at the rear end, and all deletions/retrievals from the list are made at the front end. This abstract data type follows a First In First Out (FIFO) order where the operations are first performed on the elements which are inserted first into the data structure.

Your task is to implement a First In First Out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (enqueue, dequeue, peek, and empty).

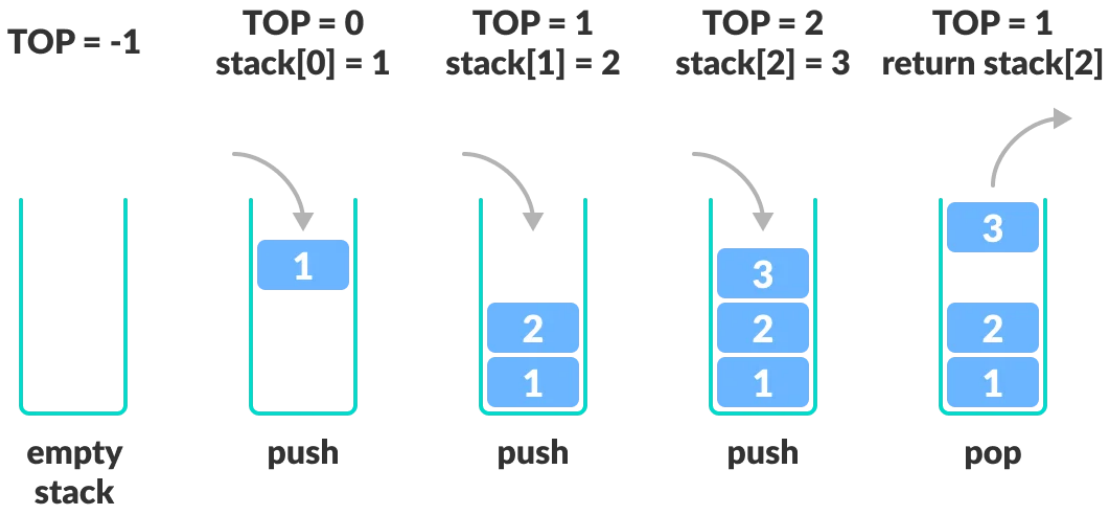
You are required to implement the following queue operations as a part of this lab exercise:

1) void enqueue(MyQueue *q, int x)
    - This function inserts an element x to the end of the queue.
2) int dequeue(MyQueue *q)
    - This function returns the element at the front of the queue and removes that element from the queue. It returns -1 if the queue is empty.
3) int peek(MyQueue *q)
    - This operation returns the element at the front of the queue. It returns -1 if the queue is empty.
4) bool empty(MyQueue *q)
    - This operation returns true if the queue is empty, else returns false.

You are supposed to use the two stacks provided to you via the boilerplate code.
These stacks are provided to you in the form of two integer arrays and their respective top variables enclosed in a structure.

```
typedef struct {
    int stack1[MAXSIZE];
    int stack2[MAXSIZE];
    int top1;
    int top2;
} MyQueue;
```

The top of the stack is initialised to -1, indicating the stack is empty.
In order to perform the PUSH operation, we just pre increment the TOP variable and store the intended element in that particular position in the stack.
Similarly for performing the POP operation, we retrieve the element present in the TOP position and simply post decrement it.
An image of these stack operations is attached below for your reference.



The stacks provided as a part of the MyQueue structure can be accessed in the following way:

```
int topElement(MyQueue *q){
    return q->stack1[q->top1];
}

int topPosition(MyQueue *q){
    return q->top1;
}
```

Note:
You are required to use the stack1 for inserting an element onto the queue.
The stack2 should only be used as an auxiliary stack and remain empty at all times.
The validate operation provided as a part of the boilerplate checks for the validity of both the stacks.
So MAKE SURE to use the expected implementation of the problem and not a workaround of the same.
USING THE BOILERPLATE CODE PROVIDED BY US IS MANDATORY.
YOU WILL FAIL MOST OF THE TEST CASES IN CASE YOU FAIL TO DO SO.

- Input format:
  Each test case starts with a digit ‘n’, followed by n operations on the queue.
  The four operations of the queue are denoted with a letter each given below:
  - E (enqueue):
    - This operation inserts an element ‘x’ at the rear end of the queue where ‘x’ is an integer followed by E.
    - It prints 1 upon a successful enqueue operation.
  - D (dequeue):
    - You need to return the integer at the front end of the queue and remove it from the queue.
    - It prints the dequeued element.
    - It returns -1 if the queue is empty.
  - V (validate):
    - This operation has already been implemented as a part of the boilerplate.
    - This operation pops ‘x’ elements from stack1 where ‘x’ is an integer followed by ‘V’. It returns the element present in the stack after ‘x’ pops. If the stack goes empty at any point of time, it just returns -1.
    - This operation also checks if the stack2 is empty.
    - The stack remains intact after the validate operation is performed.
  - P (peek):
    - This operation returns the element present at the front end of the queue. It returns -1 if the queue is empty.

- Output format:
  You don’t need to print anything as a part of this lab exercise.
  Make sure to remove all the printfs and scanfs used for debugging.

- Sample test cases:
  1. TC #1:
     - Input:
       4
       E 1
       P
       V 0
       D
     - Output: 1 1 1 1

2. TC #2:
    - Input:
      6
      E 1
      E 2
      V 1
      D
      D
      D
    - Output: 1 1 1 1 2 -1