LABORATORY PROGRAM - 1

1. Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Example 1:

Input

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Explanation

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

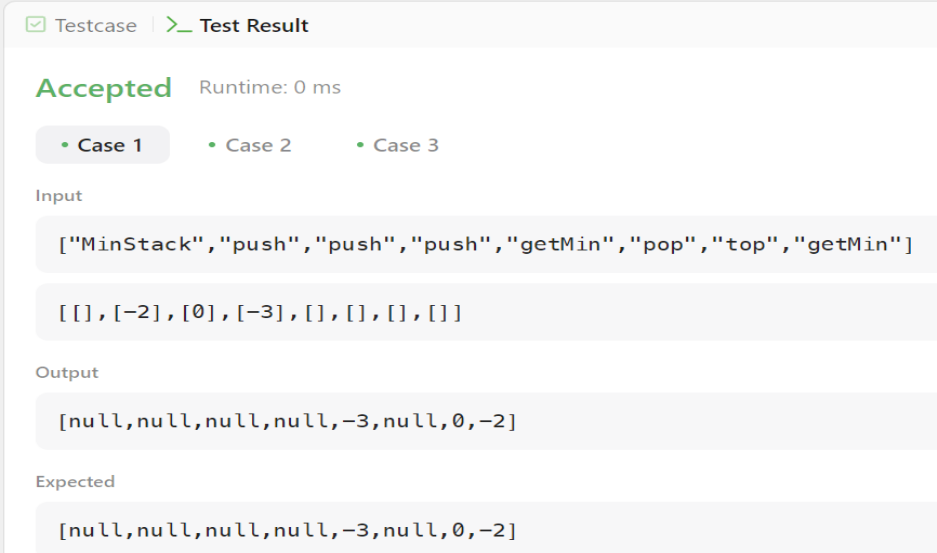
minStack.getMin(); // return -2

Constraints:

- -2^31 <= val <= 2^31 - 1
- Methods pop, top and getMin operations will always be called on non-empty stacks.
- At most 3 * 104 calls will be made to push, pop, top, and getMin.

Code

```
class MinStack {

    private Node head;

    public void push(int val) {
        if (head == null) {
            head = new Node(val, val, null);
        } else {
            head = new Node(val, Math.min(head.min, val), head);
        }
    }

    public void pop() {
        if (head != null) {
            head = head.next;
        }
    }

    public int top() {
        return head.val;
    }

    public int getMin() {
        return head.min;
    }

    private class Node {
        private int val;
        private int min;
        private Node next;

        public Node(int val, int min, Node next) {
            this.val = val;
            this.min = min;
            this.next = next;
        }
    }
}
```



2. Design Circular Deque

Design your implementation of the circular double-ended queue (deque).

Implement the MyCircularDeque class:

- MyCircularDeque(int k) Initializes the deque with a maximum size of k.
- boolean insertFront() Adds an item at the front of Deque. Returns true if the operation is successful, or false otherwise.
- boolean insertLast() Adds an item at the rear of Deque. Returns true if the operation is successful, or false otherwise.
- boolean deleteFront() Deletes an item from the front of Deque. Returns true if the operation is successful, or false otherwise.
- boolean deleteLast() Deletes an item from the rear of Deque. Returns true if the operation is successful, or false otherwise.
- int getFront() Returns the front item from the Deque. Returns -1 if the deque is empty.
- int getRear() Returns the last item from Deque. Returns -1 if the deque is empty.
- boolean isEmpty() Returns true if the deque is empty, or false otherwise.
- boolean isFull() Returns true if the deque is full, or false otherwise.

Example 1:

Input

["MyCircularDeque", "insertLast", "insertLast", "insertFront", "insertFront", "getRear", "isFull", "deleteLast", "insertFront", "getFront"]

[[3], [1], [2], [3], [4], [], [], [], [4], []]

Output

[null, true, true, true, false, 2, true, true, true, 4]

Explanation

MyCircularDeque myCircularDeque = new MyCircularDeque(3);

myCircularDeque.insertLast(1); // return True

myCircularDeque.insertLast(2); // return True

myCircularDeque.insertFront(3); // return True

myCircularDeque.insertFront(4); // return False, the queue is full.

myCircularDeque.getRear(); // return 2

myCircularDeque.isFull(); // return True

myCircularDeque.deleteLast(); // return True

myCircularDeque.insertFront(4); // return True

myCircularDeque.getFront(); // return 4

Constraints:

- 1 <= k <= 1000
- 0 <= value <= 1000
- At most 2000 calls will be made to insertFront, insertLast, deleteFront, deleteLast, getFront, getRear, isEmpty, isFull.

Code

```
class MyCircularDeque {
    private int head, tail, size, n;
    private final int[] a;

    public MyCircularDeque(int k) {
        a=new int[k];
        n=a.length;
        tail=1;

    }

    public boolean insertFront(int value) {
        if(size==n) return false;
        a[head=++head%n]=value;
        size++;
        return true;
    }

    public boolean insertLast(int value) {
        if(size==n) return false;
        a[tail =(--tail+n)%n]=value;
        size++;
        return true;
    }
```

```
    public boolean deleteFront() {
        if(size==0) return false;
        head=(--head+n)%n;
        size--;
        return true;

    }

    public boolean deleteLast() {
        if(size==0) return false;
        tail=++tail%n;
        size--;
        return true;
    }

    public int getFront() {
        return size == 0 ? -1 : a[head];
    }

    public int getRear() {
        return size == 0 ? -1 : a[tail];
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size==a.length;
    }
}
```