LeetCode Programs- Kumud Raj Ghimire

1653. Minimum Deletions to Make String Balanced

```
class Solution {
    public int minimumDeletions(String s) {
        Stack<Character> stk = new Stack<>();
        stk.push(s.charAt(0));
        int c = 0;
        for(int i=1;i<s.length();i++){
            if(!stk.isEmpty() && stk.peek() == 'b' && s.charAt(i) == 'a'){
                    stk.pop();
                    c++;
                  }
            else{
                stk.push(s.charAt(i));
            }
        }
        return c;
    }
}
```

622. Design Circular Queue

```
class MyCircularQueue {

    int front;
    int rear;
    int[] arr;
    int SIZE;

    public int next(int i){
        return (i+1)%SIZE;
    }
    public int prev(int i){
        return (i+SIZE-1)%SIZE;
    }


    public MyCircularQueue(int k) {
        arr = new int[k];
        SIZE=k;
        front=-1;
```

```
    rear=-1;
}

public boolean enQueue(int value) {
    if(isFull())return false;
    if(front==-1){
        front=0;
        rear=0;
        arr[rear]=value;
        return true;
    }
    rear = next(rear);
    arr[rear]=value;
    return true;
}

public boolean deQueue() {
    if(isEmpty())return false;
    if(front==rear){
        front=-1;
        rear=-1;
        return true;
    }
    front=next(front);
    return true;
}

public int Front() {
    if(front==-1)return -1;
    return arr[front];
}

public int Rear() {
    if(rear==-1)return -1;
    return arr[rear];
}

public boolean isEmpty() {
    return front==-1;
}

public boolean isFull() {
    return front!=-1 && next(rear)==front;
}
```

```
}
```

2390. Removing Stars From a String

```
class Solution {
    public String removeStars(String s) {
        Stack <Character> stk = new Stack<>();
        char c;
        StringBuffer sb = new StringBuffer();

        for(int i=0; i<s.length();i++){
            c = s.charAt(i);
            if(c!='*'){
                stk.push(c);
            }
            else{
                stk.pop();
            }
        }

        while(!stk.isEmpty()){
            sb.append(stk.pop());
        }

        return sb.reverse().toString();

    }
}
```

150. Evaluate Reverse Polish Notation

```
class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stk = new Stack<>();
        int len = tokens.length;
        int op1, op2;
        int result = 0;

        for (int i = 0; i < len; i++) {
            String token = tokens[i];
```

```
            if (token.equals("+") || token.equals("-") || token.equals("*") || token.equals("/")) {
                op1 = stk.pop();
                op2 = stk.pop();

                switch (token) {
                    case "+":
                        result = op1 + op2;
                        break;
                    case "-":
                        result = op2 - op1;
                        break;
                    case "*":
                        result = op1 * op2;
                        break;
                    case "/":
                        result = op2 / op1;
                        break;
                }
                stk.push(result);
            } else {
                stk.push(Integer.parseInt(token));  // Handle numbers
            }
        }
        return stk.pop();
    }
}
```

1823. Find the Winner of the Circular Game

```
class Solution {
    public int findTheWinner(int n, int k) {
        int winner=0;
        for (int i = 1; i <= n; i++) {
            winner = (winner + k) % i;
        }
        return winner + 1;
    }
}
```

225. Implement Stack using Queues

```
import java.util.LinkedList;
import java.util.Queue;
```

```
class MyStack {
    private Queue<Integer> queue1;
    private Queue<Integer> queue2;

    public MyStack() {
        queue1 = new LinkedList<>();
        queue2 = new LinkedList<>();
    }


    public void push(int x) {
        queue1.offer(x);
    }

    public int pop() {
        while (queue1.size() > 1) {
            queue2.offer(queue1.poll());
        }
        int topElement = queue1.poll();


        Queue<Integer> temp = queue1;
        queue1 = queue2;
        queue2 = temp;

        return topElement;
    }


    public int top() {
        while (queue1.size() > 1) {
            queue2.offer(queue1.poll());
        }

        int topElement = queue1.peek();

        queue2.offer(queue1.poll());


        Queue<Integer> temp = queue1;
        queue1 = queue2;
        queue2 = temp;

        return topElement;
```

```
    }

    public boolean empty() {
        return queue1.isEmpty();
    }
}
```

147. Insertion Sort List

```
class Solution {
    public ListNode insertionSortList(ListNode head) {
        ListNode dummy = new ListNode(9999);
        ListNode current = head;

        while(current!=null){
            ListNode prev=dummy;
            ListNode nextNode=current.next;
            while(prev.next!=null && prev.next.val < current.val){
                prev=prev.next;
            }
            current.next=prev.next;
            prev.next=current;
            current=nextNode;
        }
        return dummy.next;
    }
}
```

148. Sort List

```
class Solution {
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode t1 = head;

        while (t1 != null) {
            ListNode t2 = t1.next;

            while (t2 != null) {
                if (t1.val > t2.val) {
                    int temp = t1.val;
```

```
                t1.val = t2.val;
                t2.val = temp;
            }
            t2 = t2.next;
        }
        t1 = t1.next;
    }

    return head;
  }
}
```

98. Validate Binary Search Tree

```
class Solution{
  public boolean isValidBST(TreeNode root) {
    ArrayList<Integer> inorderList = new ArrayList<>();

    inorderTraversal(root, inorderList);

    for(int i=1;i<inorderList.size();i++){
      if(inorderList.get(i-1)>=inorderList.get(i))
        return false;
    }
    return true;
  }

  void inorderTraversal(TreeNode node, ArrayList<Integer> list) {
    if (node == null) {
      return;
    }

    inorderTraversal(node.left, list);

    list.add(node.val);

    inorderTraversal(node.right, list);
  }
}
```

114. Flatten Binary Tree to Linked List

```
class Solution {
```

```
    public void flatten(TreeNode root) {
        if(root==null)
        return;
        Stack <TreeNode> stk = new Stack<TreeNode>();
        stk.push(root);
        while(!stk.isEmpty()){
            TreeNode cur = stk.peek();
            stk.pop();

            if(cur.right!=null)
                stk.push(cur.right);
            if(cur.left!=null)
                stk.push(cur.left);
            if(!stk.isEmpty())
                cur.right=stk.peek();
            cur.left=null;
        }
    }
}
```

116. Populating Next Right Pointers in Each Node

```
class Solution {
    public Node connect(Node root) {
        if(root==null) return root;

        if(root.left!=null)
        root.left.next=root.right;

        if(root.right!=null && root.next!=null)
        root.right.next = root.next.left;

        connect(root.left);
        connect(root.right);

        return root;
    }
}
```

230. Kth Smallest Element in a BST

```
class Solution {
    public int kthSmallest(TreeNode root, int k) {
        ArrayList<Integer> inorderList = new ArrayList<>();
```

```
        inorderTraversal(root, inorderList);

        return inorderList.get(k-1);


    }

    void inorderTraversal(TreeNode node, ArrayList<Integer> list) {
        if (node == null) {
            return;
        }

        inorderTraversal(node.left, list);

        list.add(node.val);

        inorderTraversal(node.right, list);
    }
}
```