**1.Valid parentheses**

bool isValid(char\* s) {

    int top=-1;

    int stck[10000];

    for(int i=0;s[i]!='\0';i++)

    {

        char cur=s[i];

        if(cur=='('||cur=='{'||cur=='[')

        {

            stck[++top]=cur;

        }

        else

        {

            if(top==-1)

            {

                return false;

            }

            char topchar=stck[top--];

            if ((cur==')'&& topchar!='(')||(cur=='}'&& topchar!='{')||(cur==']'&& topchar!='['))

            {

                return false;

            }

        }

    }

    return top==-1;

}

**2. Implement Queue using Stacks**

#include <stack>

class MyQueue {

public:

    std::stack<int> stck1;

    std::stack<int> stck2;

    MyQueue() {

    }

    void push(int x) {

        stck1.push(x);

    }

    int pop() {

        if (stck2.empty()) {

            while (!stck1.empty()) {

                stck2.push(stck1.top());

                stck1.pop();

            }

        }

        int tp = stck2.top();

        stck2.pop();

        return tp;

    }

    int peek() {

        if (stck2.empty()) {

            while (!stck1.empty()) {

                stck2.push(stck1.top());

                stck1.pop();

            }

        }

        return stck2.top();

    }

    bool empty() {

        return stck1.empty() && stck2.empty();

    }

};

/\*\*

 \* Your MyQueue object will be instantiated and called as such:

 \* MyQueue\* obj = new MyQueue();

 \* obj->push(x);

 \* int param\_2 = obj->pop();

 \* int param\_3 = obj->peek();

 \* bool param\_4 = obj->empty();

 \*/

**3. Valid Parenthesis is String**

class Solution {

public:

    bool checkValidString(string s) {

        int leftMin = 0, leftMax = 0;

        for (char c : s) {

            if (c == '(') {

                leftMin++;

                leftMax++;

            } else if (c == ')') {

                leftMin--;

                leftMax--;

            } else {

                leftMin--;

                leftMax++;

            }

            if (leftMax < 0) return false;

            if (leftMin < 0) leftMin = 0;

        }

        return leftMin == 0;

    }

};

**4. Minimum Deletion to make string balanced**

class Solution {

    public int minimumDeletions(String s) {

        int aCount = 0;

        for (char c : s.toCharArray()) {

            if (c == 'a') aCount++;

        }

        int minDeletions = aCount;

        int bCount = 0;

        for (char c : s.toCharArray()) {

            if (c == 'b') {

                bCount++;

            } else {

                aCount--;

            }

            minDeletions = Math.min(minDeletions, bCount + aCount);

        }

        return minDeletions;

    }

}

**5. Minimum Twin Sum of Linked List**

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public int pairSum(ListNode head) {

        int ans=0;

        ListNode newlist=null;

        ListNode current=head;

        ListNode fast=head;

        while(fast!=null&&fast.next!=null){

            fast=fast.next.next;

            ListNode temp=current.next;

            current.next=newlist;

            newlist=current;

            current=temp;

        }

        while(current!=null){

            ans=Math.max(ans,current.val+newlist.val);

            current=current.next;

            newlist=newlist.next;

        }

        return ans;

    }

}

**6. Removing Stars From a String**

class Solution {

public:

    string removeStars(string s) {

        int n = s.size();

        int writeIndex = 0;

        for (int i = 0; i < n; ++i) {

            if (s[i] == '\*') {

                if (writeIndex > 0) {

                    --writeIndex;

                }

            } else {

                s[writeIndex] = s[i];

                ++writeIndex;

            }

        }

        return s.substr(0, writeIndex);

    }

};

**Remove Nodes from Linked list**

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode removeNodes(ListNode head) {

        ListNode current,temp;

        current=null;

        while(head!=null){

            temp=head;

            head=head.next;

            temp.next=current;

            current=temp;

        }

        head=current;

        ListNode maxNode=head;

        while(current!=null&&current.next!=null){

            if(current.next.val<maxNode.val){

                current.next=current.next.next;

            }else{

                current=current.next;

                maxNode=current;

            }

        }

        current=null;

        while(head!=null){

            temp=head;

            head=head.next;

            temp.next=current;

            current=temp;

        }

        return current;

    }

}

**8. Maximum Sum Circular Subarray**

class Solution {

    public int maxSubarraySumCircular(int[] nums) {

      int n=nums.length;

      int maxsum=nums[0],minsum=nums[0],curmax=0,curmin=0;

      int totalsum=0;

      for(int num:nums)

      {

        curmax=Math.max(curmax+num,num);

        maxsum=Math.max(maxsum,curmax);

        curmin=Math.min(curmin+num,num);

        minsum=Math.min(minsum,curmin);

        totalsum+=num;

      }

      return maxsum>0? (Math.max(maxsum,totalsum-minsum)):maxsum;

    }

}

**9. Design Front Middle Back Queue**

class FrontMiddleBackQueue {

private:

    std::deque<int> que;

public:

    FrontMiddleBackQueue() {

    }

    void pushFront(int val) {

        que.push\_front(val);

    }

    void pushMiddle(int val) {

        int midIndex = que.size() / 2;

        que.insert(que.begin() + midIndex, val);

    }

    void pushBack(int val) {

        que.push\_back(val);

    }

    int popFront() {

        if (que.empty())

        return -1;

        int frontVal = que.front();

        que.pop\_front();

        return frontVal;

    }

    int popMiddle() {

        if(que.empty())

        return -1;

        int mi=(que.size()-1)/2;

        int midVal = que[mi];

        que.erase(que.begin() + mi);

        return midVal;

    }

    int popBack() {

        if(que.empty())

        return -1;

        int val=que.back();

        que.pop\_back();

        return val;

    }

};

/\*\*

 \* Your FrontMiddleBackQueue object will be instantiated and called as such:

 \* FrontMiddleBackQueue\* obj = new FrontMiddleBackQueue();

 \* obj->pushFront(val);

 \* obj->pushMiddle(val);

 \* obj->pushBack(val);

 \* int param\_4 = obj->popFront();

 \* int param\_5 = obj->popMiddle();

 \* int param\_6 = obj->popBack();

 \*/

**10. Find the Winner of Circular Game**

class Solution {

    public int findTheWinner(int n, int k) {

        ArrayList<Integer> cir=new ArrayList<>();

        for(int i=1;i<=n;i++)

        {

            cir.add(i);

        }

        int cur=0,next;

        while(cir.size()>1)

        {

            next=(cur+k-1)%cir.size();

            cir.remove(next);

            cur=(next)%cir.size();

        }

        return cir.get(0);

    }

}

**11. Continuous Subarrays**

class Solution {

    public long continuousSubarrays(int[] nums) {

       Deque<Integer> minDeque = new LinkedList<>();

        Deque<Integer> maxDeque = new LinkedList<>();

        long count = 0;

        int l = 0;

        for (int r = 0; r < nums.length; r++) {

            while (!maxDeque.isEmpty() && nums[maxDeque.peekLast()] <= nums[r]) {

                maxDeque.pollLast();

            }

            maxDeque.offerLast(r);

            while (!minDeque.isEmpty() && nums[minDeque.peekLast()] >= nums[r]) {

                minDeque.pollLast();

            }

            minDeque.offerLast(r);

            while (nums[maxDeque.peekFirst()] - nums[minDeque.peekFirst()] > 2) {

                if (maxDeque.peekFirst() == l) maxDeque.pollFirst();

                if (minDeque.peekFirst() == l) minDeque.pollFirst();

                l++;

            }

            count += (r - l + 1);

        }

        return count;

    }

}

**12.** **Minimum Operations to Make Binary Array Elements Equal to One I**

class Solution {

    public int minOperations(int[] nums) {

        int count=0;

        int n=nums.length;

        for(int i=0;i<n-2;i++){

            if(nums[i]==0){

                nums[i]^=1;

                nums[i+1]^=1;

                nums[i+2]^=1;

                count++;

            }

        }

        if(nums[n-1]==0||nums[n-2]==0){

            return -1;

        }

        return count;

    }

}

**13.** **Swapping Nodes in a Linked List**

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode swapNodes(ListNode head, int k) {

        int a[]=new int [100000];

        int i=0,n=0;

        ListNode current= head;

        while(current!=null){

            a[i]=current.val;

            i++;

            n++;

            current=current.next;

        }

        int temp=a[k-1];

        a[k-1]=a[n-k];

        a[n-k]=temp;

        current=head;

        i=0;

        while(current!=null){

            current.val=a[i];

            i++;

            current=current.next;

        }

        return head;

    }

}

**14. Remove Nodes from Linked List**

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode removeNodes(ListNode head) {

        ListNode current,temp;

        current=null;

        while(head!=null){

            temp=head;

            head=head.next;

            temp.next=current;

            current=temp;

        }

        head=current;

        ListNode maxNode=head;

        while(current!=null&&current.next!=null){

            if(current.next.val<maxNode.val){

                current.next=current.next.next;

            }else{

                current=current.next;

                maxNode=current;

            }

        }

        current=null;

        while(head!=null){

            temp=head;

            head=head.next;

            temp.next=current;

            current=temp;

        }

        return current;

    }

}