LeetCode stack question with its **solution**:-

Leetcode problem *#20* - Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

Example 4:

Input: s = "([])"

Output: true

Solution-

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for (char ch : s) {
            if (ch == '(' || ch == '[' || ch == '{') {
                st.push(ch);
            } else {
                if (st.empty()) return false;
                char top = st.top();
                if ((ch == ')' && top != '(') ||
                    (ch == ']' && top != '[') ||
                    (ch == '}' && top != '{')) {
                    return false;
                }
                st.pop();
            }
        }
        return st.empty();
    }
};
```

Leetcode problem *#155* - 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

Solution-

```
class MinStack {
private:
    stack<int> s;
    stack<int> minS;

public:
    MinStack() {
    }

    void push(int val) {
        s.push(val);
        if (minS.empty() || val <= minS.top()) {
            minS.push(val);
        } else {
            minS.push(minS.top());
```

```
        }
    }

    void pop() {
        if (!s.empty()) {
            s.pop();
            minS.pop();
        }
    }

    int top() {
        return s.top();
    }

    int getMin() {
        return minS.top();
    }
};
```

Leetcode problem #*503* - Next Greater Element ||

Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return *the* ***next greater number*** *for every element in* nums.

The **next greater number** of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number.

Example 1:

Input: nums = [1,2,1]
Output: [2,-1,2]
Explanation: The first 1's next greater number is 2;
The number 2 can't find next greater number.
The second 1's next greater number needs to search circularly, which is also 2.

Example 2:

Input: nums = [1,2,3,4,3]
Output: [2,3,4,-1,4]

Solution-

```
class Solution {
public:
    vector<int> nextGreaterElements(vector<int>& nums) {
        int n = nums.size();
        vector<int> res(n, -1);
        stack<int> st;

        for (int i = 2 * n - 1; i >= 0; i--) {
            while (!st.empty() && st.top() <= nums[i % n]) {
                st.pop();
            }
            if (i < n) {
                if (!st.empty()) res[i] = st.top();
            }
            st.push(nums[i % n]);
        }

        return res;
    }
};
```

Leetcode problem *#682* - Baseball Game

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the ith operation you must apply to the record and is one of the following:

- An integer x.
    - Record a new score of x.
- '+'.
    - Record a new score that is the sum of the previous two scores.
- 'D'.

- Record a new score that is the double of the previous score.
- 'C'.
  - Invalidate the previous score, removing it from the record.

Return *the sum of all the scores on the record after applying all the operations*.

The test cases are generated such that the answer and all intermediate calculations fit in a **32-bit** integer and that all operations are valid.

Example 1:

Input: ops = ["5","2","C","D","+"]
Output: 30
Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"C" - Invalidate and remove the previous score, record is now [5].
"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
The total sum is 5 + 10 + 15 = 30.

Example 2:

Input: ops = ["5","-2","4","C","D","9","+","+"]
Output: 27
Explanation:
"5" - Add 5 to the record, record is now [5].
"-2" - Add -2 to the record, record is now [5, -2].
"4" - Add 4 to the record, record is now [5, -2, 4].
"C" - Invalidate and remove the previous score, record is now [5, -2].
"D" - Add 2 * -2 = -4 to the record, record is now [5, -2, -4].
"9" - Add 9 to the record, record is now [5, -2, -4, 9].
"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].
"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].
The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.

Example 3:

Input: ops = ["1","C"]
Output: 0
Explanation:

"1" - Add 1 to the record, record is now [1].
"C" - Invalidate and remove the previous score, record is now [].
Since the record is empty, the total sum is 0.

Solution-

```
class Solution {
public:
    int calPoints(vector<string>& operations) {
        stack<int> st;

        for (string op : operations) {
            if (op == "C") {
                st.pop();
            } else if (op == "D") {
                st.push(2 * st.top());
            } else if (op == "+") {
                int top1 = st.top(); st.pop();
                int top2 = st.top();
                st.push(top1);
                st.push(top1 + top2);
            } else {
                st.push(stoi(op));
            }
        }

        int total = 0;
        while (!st.empty()) {
            total += st.top();
            st.pop();
        }
        return total;
    }
};
```

Leetcode problem *#1047* - Remove All Adjacent Duplicates In String

You are given a string s consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on s until we no longer can.

Return *the final string after all such duplicate removals have been made*. It can be proven that the answer is **unique**.

Example 1:

Input: s = "abbaca"
Output: "ca"
Explanation:
For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

Example 2:

Input: s = "azxxzy"
Output: "ay"

Solution-

```
class Solution {
public:
    string removeDuplicates(string s) {
        stack<char> st;

        for (char c : s) {
            if (!st.empty() && st.top() == c) {
                st.pop(); // remove duplicate
            } else {
                st.push(c); // add character
            }
        }
```

```
        // Build final string from stack
        string result = "";
        while (!st.empty()) {
            result += st.top();
            st.pop();
        }
        reverse(result.begin(), result.end());
        return result;
    }
};
```