Day 35 / 100 :

Topic - Strings

1 Problem statement: Removing star from string (Medium)

You are given a string s, which contains stars *.
In one operation, you can:

- Choose a star in s.
- Remove the closest non-star character to its left, as well as remove the star itself.

Return the string after all stars have been removed.
Note:

- The input will be generated such that the operation is always possible.
- It can be shown that the resulting string will always be unique.

Example 1:
Input: s = "leet**cod*e"
Output: "lecoe"
Explanation: Performing the removals from left to right:
- The closest character to the 1st star is 't' in "lee**t****cod*e". s becomes "lee*cod*e".
- The closest character to the 2nd star is 'e' in "le**e***cod*e". s becomes "lecod*e".
- The closest character to the 3rd star is 'd' in "leco**d***e". s becomes "lecoe".

There are no more stars, so we return "lecoe".

Example 2:
Input: s = "erase*****"
Output: ""
Explanation: The entire string is removed, so we return an empty string

Solutions :

Approach 1 - Two pointers

- Start with two pointers from the first element(i representing a traversal and string from 0 to j-1 represents answer till ith element)
- If you find a non * element, increase j as it'll be part of answer.
- But if you find * then remove the last element of our answer(i.e. decrease j).
- After traversing all the elements return substring till j as answer.(refer point 1).

```cpp
class Solution {
public:
    string removeStars(string s) {
        int i=0,j=0;
        for(i=0;i<s.size();i++){
            if(s[i]=='*'){
                j--;
            }else{
                s[j++] = s[i];
            }
        }
        return s.substr(0,j);
    }
};
```

Solutions :

Approach 2 - Using Stack

- Since the question says if you find * remove the left closest element and this operation is always possible (i.e. we don't need to worry about the conditions)
- Start pushing element in stack, as soon as you find * pop the top element from stack(There will be always atleast an element in stack since this operation is always possible).
- Store all the elements of stack in a variable.
- Since we have performed this operation in stack therfore the obtained string will be in reverse order.
- Reverse the string and return it as answer

Note: The different idea of reversing the string may give you TLE.

- The reason why using ans.push_back(st.top()) and reversing the string before returning it works fine, while using ans = st.top() + ans results in TLE, is because the + operator for strings is an expensive operation.
- When you use ans = st.top() + ans, for every iteration of the while loop, you are creating a new string by concatenating the current character with the existing string. This new string is then assigned to the ans variable. As the size of the string increases, this operation becomes increasingly expensive, resulting in a TLE for larger inputs.

(The part inside note is contributed by @cenacr007_harsh in comment section.)

```cpp
class Solution {
public:
    string removeStars(string s) {
        stack<char> stk;
        for(int i=0;i<s.size();i++){
            char cur = s[i];
            if(cur == '*'){
                stk.pop();
            }else{
                stk.push(cur);
            }
        }
        string ans = "";
        while(!stk.empty()){
            ans += stk.top();
            stk.pop();
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }
};
```

2 Problem statement: Rotate Array (Medium)

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

Example 1:
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]

Example 2:
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]

Solutions :

Approach 1 - Using Stack

here i is of loop which starts form (0,1,2...) k is the value how many times you want to rotate and n is the size of first vector ,and after that new indexes will be generated in our temp vector and we have to put values of nums vector at new indexes . . .
like for first arr[1,2,3,4,5,6,7] and k=3 so ,
temp[0 + 3 % 7]=nums[i]
temp[3 % 7]=nums[i]
temp[3] = nums[i]

```
[_ , _ , _ , 1 , _ ,_ , _ ]
```

for next ,
temp[1 + 3 % 7]=nums[i]
temp[4 % 7]=nums[i]
temp[4] = nums[i]

```
[_ , _ , _ , 1 , 2 ,_ , _ ]
```

and so on…

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {

     int n=nums.size();
        vector<int> temp(nums.size());
        for(int i=0;i<n;i++){
            temp[(i+k)%n]=nums[i];
        }
        nums=temp;
    }
};
```