DSA SERIES

- Learn Coding

Topic to be Covered today

Stack

LETS START TODAY'S LECTURE

Longest Valid Parentheses (32)

```
class Solution {
public:
    int longestValidParentheses(string s) {
        int L = 0;
        int R = 0;
        int Max = 0;

        //  Left to right pass

        for (int i = 0; i < s.length(); i++) {
            if (s[i] == '(') {
                L++;
            } else {
                R++;
            }

            if (L == R) {
                Max = max(Max, 2 * L);
            } else if (R > L) {
                L = 0;
                R = 0;
```

```
        }
        }
        // Right to Left
        L = 0;
        R = 0;

        for (int i = s.length() - 1; i >= 0; i--) {
            if (s[i] == '(') {
                L++;
            } else {
                R++;
            }

            if (L == R) {
                Max = max(Max, 2 * L);
            } else if (L > R) {
                L = 0;
                R = 0;
            }
        }
        return Max;
    }
};
```

Using Stack

```
class Solution {
public:
    int longestValidParentheses(string s) {
        stack<int> st;
        st.push(-1);
        int maxLen  = 0;

        for(int i = 0;i<s.length();i++){
            if(s[i]=='('){
                st.push(i);
            }else{
                st.pop();

                if(st.empty()){
                    st.push(i);
                } else {
                    maxLen = max(maxLen , i - st.top());
                }
            }
        }
        return maxLen;
    }
};
```

Min Stack (155)

```
class MinStack {
public:
    vector<vector<int>> st;
    MinStack() {}

    void push(int val) {
        int min = getMin();
        if (st.empty() || min > val) {
            min = val;
        }

        st.push_back({val, min});
    }

    void pop() { st.pop_back(); }

    int top() {
        return st.empty() ? -1 : st.back()[0];
        }

    int getMin() {
        return st.empty() ? -1 : st.back()[1];
    }
};
```

Maximum Frequency Stack (895)

```
class FreqStack {
public:
    unordered_map<int,int> freq;
    unordered_map<int,stack<int>> freqMap ;
    int maxFreq  = 0;

    FreqStack() {

    }

    void push(int val) {
        freq[val]++;

        int f = freq[val];
        freqMap[f].push(val);

        if(f > maxFreq){
            maxFreq = f;
        }

    }
```

```
    int pop() {

        int val = freqMap[maxFreq].top();
        freqMap[maxFreq].pop();

        freq[val]--;

        if(freqMap[maxFreq].empty()){
            maxFreq--;
        }
        return val;

    }
};
```

Design a stack with Increment Operation (1381)

```
class CustomStack {
public:
    vector<int> st;
    int top;
    int N;

    CustomStack(int maxSize) {
        N = maxSize;
        st.resize(N);
        top = -1;
    }

    void push(int x) {
        if (top == N - 1) {
            return;
        }

        top++;
        st[top] = x;

        // st[++top]=x;
    }
```

```
int pop() {

        if (top == -1) {
            return -1;
        }

        return st[top--];
    }

    void increment(int k, int val) {

        int n = st.size();

        if (n < k) {
            for (int i = 0; i < n; i++) {
                st[i] = st[i] + val;
            }
        } else if (st.size() >= k) {
            for (int i = 0; i < k; i++) {
                st[i] = st[i] + val;
            }
        }
    }
};
```

Learn coding

THANK YOU