DAY-4

Name- Maadhav Hira **UID-** 22BCS10380 **Group-** KPIT-901(A)

1.VERY EASY:

1 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.

Code:-

```
#include <stack>
#include <iostream>
#include <string>
using namespace std;

class MinStack {
private:
    stack<int> s; // Stack to store all elements
    stack<int> minStack; // Stack to store the minimum values

public:
    MinStack() {
        // Initialize the stack
    }

    // Pushes an element onto the stack
    void push(int val) {
        s.push(val);
        // Push to minStack the smaller value between val and the current minimum
        if (minStack.empty() || val <= minStack.top()) {
```

```
            minStack.push(val);
        }
    }

    // Removes the element on the top of the stack
    void pop() {
        if (!s.empty()) {
            if (s.top() == minStack.top()) {
                minStack.pop(); // Pop from minStack if it's the current minimum
            }
            s.pop();
        }
    }

    // Gets the top element of the stack
    int top() {
        if (!s.empty()) {
            return s.top();
        }
        throw runtime_error("Stack is empty.");
    }

    // Retrieves the minimum element in the stack
    int getMin() {
        if (!minStack.empty()) {
            return minStack.top();
        }
        throw runtime_error("Stack is empty.");
    }
};

int main() {
    MinStack minStack;
    string command;
    int value;

    cout << "Enter commands (push <value>, pop, top, getMin, exit):" << endl;

    while (true) {
        cin >> command;
        if (command == "push") {
```

```
            cin >> value;
            minStack.push(value);
            cout << "Pushed " << value << " onto the stack." << endl;
        } else if (command == "pop") {
            try {
                minStack.pop();
                cout << "Popped the top element from the stack." << endl;
            } catch (exception &e) {
                cout << "Error: " << e.what() << endl;
            }
        } else if (command == "top") {
            try {
                cout << "Top element is: " << minStack.top() << endl;
            } catch (exception &e) {
                cout << "Error: " << e.what() << endl;
            }
        } else if (command == "getMin") {
            try {
                cout << "Minimum element is: " << minStack.getMin() << endl;
            } catch (exception &e) {
                cout << "Error: " << e.what() << endl;
            }
        } else if (command == "exit") {
            cout << "Exiting program." << endl;
            break;
        } else {
            cout << "Invalid command. Please try again." << endl;
        }
    }

    return 0;
}
```

2. Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

Code:-

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
```

```
int firstUniqChar(string s) {
    vector<int> freq(26, 0); // Array to store frequency of each character

    // First traversal: Count the frequency of each character
    for (char c : s) {
        freq[c - 'a']++;
    }

    // Second traversal: Find the first character with frequency 1
    for (int i = 0; i < s.length(); i++) {
        if (freq[s[i] - 'a'] == 1) {
            return i; // Return the index of the first non-repeating character
        }
    }

    return -1; // No non-repeating character found
}

int main() {
    string s;
    cout << "Enter the string: ";
    cin >> s;

    int result = firstUniqChar(s);

    if (result != -1) {
        cout << "The index of the first non-repeating character is: " << result
<< endl;
    } else {
        cout << "No non-repeating character exists." << endl;
    }

    return 0;
}
```

3. Implement a simple text editor. The editor initially contains an empty string, S.Perform Q operations of the following 4 types:

- append(W) - Append string W to the end of S.
- delete (k)- Delete the last k characters of S.

- print (k)- Print the k^th character of S.
- undo() - Undo the last (not previously undone) operation of type 1 or 2, reverting S to the state it was in prior to that operation.

Code:-

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

class TextEditor {
private:
    string S; // The main text
    stack<pair<int, string>> history; // Stack to store history for undo

public:
    // Append W to the string
    void append(const string& W) {
        history.push({1, W}); // Save operation type and appended string
        S += W;
    }

    // Delete the last k characters
    void deleteChars(int k) {
        string deleted = S.substr(S.size() - k); // Capture the deleted part
        history.push({2, deleted}); // Save operation type and deleted string
        S.erase(S.size() - k);
    }

    // Print the k-th character
    void print(int k) const {
        if (k > 0 && k <= S.size()) {
            cout << S[k - 1] << endl; // 1-based index
        }
    }

    // Undo the last append or delete operation
    void undo() {
        if (!history.empty()) {
            auto lastOp = history.top();
            history.pop();
```

```
            if (lastOp.first == 1) {
                // Undo append
                int len = lastOp.second.size();
                S.erase(S.size() - len);
            } else if (lastOp.first == 2) {
                // Undo delete
                S += lastOp.second;
            }
        }
    }
};

int main() {
    int Q;
    cout << "Enter the number of operations: ";
    cin >> Q;

    TextEditor editor;

    for (int i = 0; i < Q; i++) {
        int type;
        cin >> type;

        if (type == 1) {
            string W;
            cin >> W;
            editor.append(W);
        } else if (type == 2) {
            int k;
            cin >> k;
            editor.deleteChars(k);
        } else if (type == 3) {
            int k;
            cin >> k;
            editor.print(k);
        } else if (type == 4) {
            editor.undo();
        }
    }

    return 0;
}
```

4. Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).
Implement the MyQueue class:
void push(int x) Pushes element x to the back of the queue.
int pop() Removes the element from the front of the queue and returns it.
int peek() Returns the element at the front of the queue.
boolean empty() Returns true if the queue is empty, false otherwise.
Notes:
You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Code:-

```
#include <iostream>
#include <stack>
using namespace std;

class MyQueue {
private:
    stack<int> inputStack;  // Stack for pushing elements
    stack<int> outputStack; // Stack for popping/peeking elements

    // Transfer elements from inputStack to outputStack
    void transfer() {
        while (!inputStack.empty()) {
            outputStack.push(inputStack.top());
            inputStack.pop();
        }
    }

public:
    // Initialize the queue
    MyQueue() {}

    // Push element x to the back of the queue
    void push(int x) {
        inputStack.push(x);
    }
```

```
    // Removes the element from the front of the queue and returns it
    int pop() {
        if (outputStack.empty()) {
            transfer();
        }
        int front = outputStack.top();
        outputStack.pop();
        return front;
    }

    // Get the front element
    int peek() {
        if (outputStack.empty()) {
            transfer();
        }
        return outputStack.top();
    }

    // Returns whether the queue is empty
    bool empty() {
        return inputStack.empty() && outputStack.empty();
    }
};

int main() {
    MyQueue myQueue;

    myQueue.push(1); // queue: [1]
    myQueue.push(2); // queue: [1, 2]
    cout << "Front element (peek): " << myQueue.peek() << endl; // return
1
    cout << "Popped element: " << myQueue.pop() << endl; // return 1,
queue: [2]
    cout << "Is queue empty? " << (myQueue.empty() ? "Yes" : "No") <<
endl; // return false

    return 0;
}
```

5. You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return an integer that represents the value of the expression.

Note that:

- The valid operators are '+', '-', '*', and '/'.
- Each operand may be an integer or another expression.
- The division between two integers always truncates toward zero.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a 32-bit integer.

Code:-

```
#include <iostream>
#include <vector>
#include <stack>
#include <string>
#include <unordered_map>
#include <functional>
using namespace std;

int evalRPN(vector<string>& tokens) {
    // Define operations using a lambda map
    unordered_map<string, function<int(int, int)>> operations = {
        {"+", [](int a, int b) { return a + b; }},
        {"-", [](int a, int b) { return a - b; }},
        {"*", [](int a, int b) { return a * b; }},
        {"/", [](int a, int b) { return a / b; }} // Integer division truncates
toward zero
    };

    stack<int> s;

    for (const string& token : tokens) {
        if (operations.count(token)) { // If the token is an operator
            int b = s.top(); s.pop();
            int a = s.top(); s.pop();
```

```
            s.push(operations[token](a, b)); // Apply the operation and push
the result
        } else { // If the token is a number
            s.push(stoi(token)); // Convert string to integer and push to stack
        }
    }

    // The final result is the only element left in the stack
    return s.top();
}

int main() {
    vector<string> tokens1 = {"2", "1", "+", "3", "*"};
    vector<string> tokens2 = {"4", "13", "5", "/", "+"};
    vector<string> tokens3 = {"10", "6", "9", "3", "+", "-11", "*", "/", "*",
"17", "+", "5", "+"};

    cout << "Example 1 Output: " << evalRPN(tokens1) << endl; // Output:
9
    cout << "Example 2 Output: " << evalRPN(tokens2) << endl; // Output:
6
    cout << "Example 3 Output: " << evalRPN(tokens3) << endl; // Output:
22

    return 0;
}
```