Name – Aryan UID – 22BCS15357 SECTION – 620(B)

VERY EASY:

1 Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

```
#include <iostream>

#include <stack>

using namespace std;


class MinStack {

private:

    stack<int> mainStack;

    stack<int> minStack;


public:

    void push(int value) {

        mainStack.push(value);

        if (minStack.empty() || value <= minStack.top()) {

            minStack.push(value);

        }

    }


    void pop() {

        if (mainStack.empty()) {

            cout << "Stack Underflow\n";

            return;

        }

        if (mainStack.top() == minStack.top()) {

            minStack.pop();

        }

        mainStack.pop();

    }


    int top() {
        if (mainStack.empty()) {
            cout << "Stack is empty\n";
            return -1;
        }
        return mainStack.top();
    }

    int getMin() {
        if (minStack.empty()) {
            cout << "Stack is empty\n";
            return -1;
        }
        return minStack.top();
    }

    bool isEmpty() {
        return mainStack.empty();
    }
};

int main() {
    MinStack minStack;

    minStack.push(5);
    minStack.push(10);
    minStack.push(3);
    minStack.push(7);

    cout << "Minimum value: " << minStack.getMin() << "\n";
    cout << "Top value: " << minStack.top() << "\n";

    minStack.pop();
    cout << "Minimum value after pop: " << minStack.getMin() << "\n";
    minStack.pop();
    cout << "Minimum value after another pop: " << minStack.getMin() << "\n";


    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.

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

int firstNonRepeatingCharacter(string s) {
    int charCount[256] = {0};

    for (char c : s) {
        charCount[c]++;
    }

    for (int i = 0; i < s.size(); i++) {
        if (charCount[s[i]] == 1) {
            return i;
        }
    }

    return -1;
}

int main() {
    string input;
    cout << "Enter a string: ";
    cin >> input;

    int index = firstNonRepeatingCharacter(input);

    if (index == -1) {
        cout << "No non-repeating character found.\n";
    } else {
        cout << "The index of the first non-repeating character is: " << index << "\n";
    }

    return 0;
}
```

3 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).

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

class QueueUsingTwoStacks {
private:
    stack<int> stack1, stack2;

public:
    void push(int value) {
        stack1.push(value);
    }

    int pop() {
        if (stack2.empty()) {
            while (!stack1.empty()) {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        if (stack2.empty()) return -1;
        int front = stack2.top();
        stack2.pop();
        return front;
    }

    int peek() {
        if (stack2.empty()) {
            while (!stack1.empty()) {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        if (stack2.empty()) return -1;
        return stack2.top();
    }
    bool empty() {
        return stack1.empty() && stack2.empty();
    }
};

int main() {
    QueueUsingTwoStacks q;

    q.push(1);
    q.push(2);
    q.push(3);

    cout << "Front element: " << q.peek() << endl;
    cout << "Popped element: " << q.pop() << endl;
    cout << "Popped element: " << q.pop() << endl;
    cout << "Queue is empty: " << (q.empty() ? "Yes" : "No") << endl;
    cout << "Popped element: " << q.pop() << endl;
    cout << "Queue is empty: " << (q.empty() ? "Yes" : "No") << endl;

    return 0;
}
```

EASY LEVEL:-

4 A bracket is considered to be any one of the following characters: (,), {, }, [, or].

```
#include <iostream>

#include <stack>

#include <string>

using namespace std;



bool isBalanced(string s) {

    stack<char> st;



    for (char c : s) {

        if (c == '(' || c == '{' || c == '[') {
            st.push(c);
        } else {
            if (st.empty()) return false;
            char top = st.top();
            if ((c == ')' && top == '(') || (c == '}' && top == '{') || (c == ']' && top == '[')) {
                st.pop();
            } else {
                return false;
            }
        }
    }

    return st.empty();
}

int main() {
    string s;
    cout << "Enter a string of brackets: ";
    cin >> s;

    if (isBalanced(s)) {
        cout << "The string is balanced.\n";
    } else {
        cout << "The string is not balanced.\n";
    }
    return 0;

}
```

5 The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

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

int countStudents(vector<int>& students, vector<int>& sandwiches) {
    queue<int> studentQueue;
    queue<int> sandwichQueue;

    for (int student : students) {
        studentQueue.push(student);
    }

    for (int sandwich : sandwiches) {
        sandwichQueue.push(sandwich);
    }

    int unableToEat = 0;

    while (!studentQueue.empty() && unableToEat < studentQueue.size()) {
        if (studentQueue.front() == sandwichQueue.front()) {
            studentQueue.pop();
            sandwichQueue.pop();
            unableToEat = 0;
        } else {
            studentQueue.push(studentQueue.front());
            studentQueue.pop();
            unableToEat++;
        }
    }

    return studentQueue.size();
}

int main() {
    vector<int> students = {1, 1, 0, 0};
    vector<int> sandwiches = {0, 1, 0, 1};

    cout << "Number of students unable to eat: " << countStudents(students,
sandwiches) << endl;
```

```
    return 0;
}
```

MEDIUM:-

6 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.

```
#include <iostream>

#include <vector>

#include <stack>

using namespace std;



vector<int> nextGreaterElements(vector<int>& nums) {

    int n = nums.size();

    vector<int> result(n, -1);

    stack<int> st;



    for (int i = 0; i < 2 * n; i++) {

        while (!st.empty() && nums[st.top()] < nums[i % n]) {

            result[st.top()] = nums[i % n];

            st.pop();

        }

        if (i < n) {

            st.push(i);

        }

    }
    return result;
}

int main() {
    vector<int> nums = {1, 2, 1};
    vector<int> result = nextGreaterElements(nums);

    cout << "Next greater elements: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

7 You are given a 0-indexed string pattern of length n consisting of the characters 'I' meaning increasing and 'D' meaning decreasing.

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

vector<int> findPermutation(string pattern) {
    int n = pattern.size();
    vector<int> result;
    stack<int> st;
    for (int i = 0; i <= n; i++) {
        st.push(i + 1);
        if (i == n || pattern[i] == 'I') {
            while (!st.empty()) {
                result.push_back(st.top());
                st.pop();
            }
        }
    }

    return result;
}

int main() {
    string pattern = "IDID";
    vector<int> result = findPermutation(pattern);

    cout << "Resultant permutation: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
```

```
}
```

HARD

8 You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.
Return the max sliding window.

```
#include <iostream>

#include <vector>

#include <deque>

using namespace std;


vector<int> maxSlidingWindow(vector<int>& nums, int k) {

    vector<int> result;

    deque<int> dq;


    for (int i = 0; i < nums.size(); i++) {

        // Remove indices that are out of the bounds of the current window

        if (!dq.empty() && dq.front() == i - k) {

            dq.pop_front();

        }


        // Remove indices whose corresponding values are less than nums[i]

        while (!dq.empty() && nums[dq.back()] < nums[i]) {

            dq.pop_back();

        }


        // Add the current index

        dq.push_back(i);
        // Append the maximum of the current window to the result
        if (i >= k - 1) {
            result.push_back(nums[dq.front()]);
        }
    }

    return result;
}

int main() {
    vector<int> nums = {1, 3, -1, -3, 5, 3, 6, 7};
    int k = 3;

    vector<int> result = maxSlidingWindow(nums, k);

    for (int maxVal : result) {
        cout << maxVal << " ";
    }

    return 0;
}
```

9 You have an infinite number of stacks arranged in a row and numbered (left to right) from 0, each of the stacks has the same maximum capacity.

```
#include <iostream>

#include <vector>

#include <deque>

using namespace std;



vector<int> maxSlidingWindow(vector<int>& nums, int k) {
vector<int> result;
deque<int> dq;

  for (int i = 0; i < nums.size(); i++) {
    // Remove indices that are out of the bounds of the current window
    if (!dq.empty() && dq.front() == i - k) {
        dq.pop_front();
    }

    // Remove indices whose corresponding values are less than nums[i]
    while (!dq.empty() && nums[dq.back()] < nums[i]) {
        dq.pop_back();
    }

    // Add the current index
    dq.push_back(i);

    // Append the maximum of the current window to the result
    if (i >= k - 1) {
        result.push_back(nums[dq.front()]);
    }
  }

  return result;
}
```

```
int main() {

    vector<int> nums = {1, 3, -1, -3, 5, 3, 6, 7};

    int k = 3;


    vector<int> result = maxSlidingWindow(nums, k);


    for (int maxVal : result) {

        cout << maxVal << " ";

    }


    return 0;

}
```

10 Suppose there is a circle. There are N petrol pumps on that circle. Petrol pumps are numbered 0 to (N-1) (both inclusive). You have two pieces of information corresponding to each of the petrol pump: (1) the amount of petrol that particular petrol pump will give, and (2) the distance from that petrol pump to the next petrol pump.

```
#include <iostream>

#include <vector>

using namespace std;


int findStartingPetrolPump(vector<int>& petrol, vector<int>& distance) {

    int n = petrol.size();

    int totalPetrol = 0, totalDistance = 0;

    int start = 0, surplus = 0;
    for (int i = 0; i < n; i++) {
        totalPetrol += petrol[i];
        totalDistance += distance[i];
        surplus += petrol[i] - distance[i];

        if (surplus < 0) {
            start = i + 1; // Reset starting point
            surplus = 0;   // Reset surplus
        }
    }

    return totalPetrol >= totalDistance ? start : -1;
}

int main() {
    vector<int> petrol = {4, 6, 7, 4};
    vector<int> distance = {6, 5, 3, 5};

    int start = findStartingPetrolPump(petrol, distance);

    if (start == -1) {
        cout << "No solution exists\n";
    } else {
        cout << "Start at petrol pump: " << start << "\n";
    }


    return 0;

}
```