Name – Subhojit Ghosh
UID – 22BCS15368
Section – 620-B

DAY – 4

**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.

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

class MinStack {
private:
    stack<int> mainStack;
    stack<int> minStack; // Keeps track of minimum values

public:
    // Constructor
    MinStack() {
        // Initialize the stacks
        while (!mainStack.empty()) mainStack.pop();
        while (!minStack.empty()) minStack.pop();
    }

    // Push value onto the stack
    void push(int val) {
        mainStack.push(val);
        // Push onto minStack if it's empty or the new value is <= current min
        if (minStack.empty() || val <= minStack.top()) {
            minStack.push(val);
        }
```

```cpp
    }

    // Pop the top value from the stack
    void pop() {
        if (mainStack.top() == minStack.top()) {
            minStack.pop(); // Pop from minStack if it's the current minimum
        }
        mainStack.pop();
    }

    // Get the top value
    int top() {
        return mainStack.top();
    }

    // Retrieve the minimum value
    int getMin() {
        return minStack.top();
    }
};

int main() {
    MinStack minStack;
    minStack.push(-2);
    minStack.push(0);
    minStack.push(-3);
    cout << "Minimum: " << minStack.getMin() << endl;
    minStack.pop();
    cout << "Top: " << minStack.top() << endl;
    cout << "Minimum: " << minStack.getMin() << endl;
    return 0;
}
```

```
Minimum: -3
Top: 0
Minimum: -2


...Program finished with exit code 0
Press ENTER to exit console.
```

**2 Given a string s, find the first non-repeating character in it and return its index. If it does**

**not exist, return -1.**

**Example 1:**

**Input:** s = "leetcode"

**Output:** 0

**Explanation:**

The character 'l' at index 0 is the first character that does not occur at any other index.

**Example 2:**

**Input:** s = "loveleetcode"

**Output:** 2

```cpp
#include <iostream>

#include <unordered_map>

#include <string>

using namespace std;


int firstUniqChar(string s) {

    unordered_map<char, int> freq; // To store frequency of characters

    for (char c : s)

        freq[c]++;


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

        if (freq[s[i]] == 1)

            return i; // Return index of the first non-repeating character

    }

    return -1; // If no non-repeating character is found
```
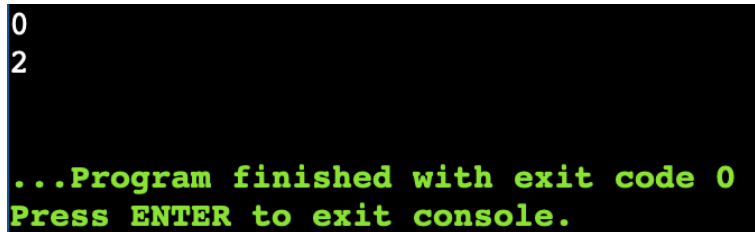
```
}


int main() {

    string s1 = "leetcode";

    string s2 = "loveleetcode";

    cout << firstUniqChar(s1) << endl; // Output: 0

    cout << firstUniqChar(s2) << endl; // Output: 2

    return 0;

}
```

```
0
2


...Program finished with exit code 0
Press ENTER to exit console.
```

**2 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.
**Example 1:**
**Input**
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
**Output**
[null, null, null, 1, 1, false]
**Explanation**
- MyQueue myQueue = new MyQueue();
- myQueue.push(1); // queue is: [1]

- myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
- myQueue.peek(); // return 1
- myQueue.pop(); // return 1, queue is [2]
- myQueue.empty(); // return false

**Example 2:**
**Input:**
 ["MyQueue", "push", "push", "push", "peek", "pop", "peek", "empty"]
[[], [3], [5], [7], [], [], [], []]
**Output:**
 [null, null, null, null, 3, 3, 5, false]


#include <stack>

#include <iostream>

using namespace std;


class MyQueue {

   stack<int> stack1, stack2; // Two stacks to simulate a queue


public:

  MyQueue() {}


  void push(int x) {

    stack1.push(x); // Push onto stack1

  }


  int pop() {

    if (stack2.empty()) {

      while (!stack1.empty()) {

```cpp
            stack2.push(stack1.top()); // Move elements from stack1 to stack2

            stack1.pop();

        }

    }

    int front = stack2.top();

    stack2.pop(); // Remove the front element

    return front;

}


int peek() {

    if (stack2.empty()) {

        while (!stack1.empty()) {

            stack2.push(stack1.top()); // Move elements from stack1 to stack2

            stack1.pop();

        }

    }

    return stack2.top(); // Return the front element

}


bool empty() {

    return stack1.empty() && stack2.empty(); // Queue is empty if both stacks are empty

}
};
```

```cpp
int main() {

    MyQueue myQueue;

    myQueue.push(1);

    myQueue.push(2);

    cout << myQueue.peek() << endl; // Output: 1

    cout << myQueue.pop() << endl;  // Output: 1

    cout << (myQueue.empty() ? "true" : "false") << endl; // Output: false


    MyQueue anotherQueue;

    anotherQueue.push(3);

    anotherQueue.push(5);

    anotherQueue.push(7);

    cout << anotherQueue.peek() << endl; // Output: 3

    cout << anotherQueue.pop() << endl;  // Output: 3

    cout << anotherQueue.peek() << endl; // Output: 5

    cout << (anotherQueue.empty() ? "true" : "false") << endl; // Output: false


    return 0;

}
```

```
1
1
false
3
3
5
false

...Program finished with exit code 0
Press ENTER to exit console.
```

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

**Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e., ), ], or }) of the exact same type. There are three types of matched pairs of brackets: [], {}, and ().**

**A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket, ].**

**By this logic, we say a sequence of brackets is balanced if the following conditions are met:**

**It contains no unmatched brackets.**

**The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.**

**Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.**

**Function Description**

Complete the function isBalanced in the editor below.

isBalanced has the following parameter(s):

string s: a string of brackets

**Returns**

string: either YES or NO

**Input Format**

The first line contains a single integer n, the number of strings.
Each of the next n lines contains a single string s, a sequence of brackets.

**Constraints**

- $1 \leq n \leq 10^3$
- $1 \leq |S| \leq 10^3$, where $|S|$ is the length of the sequence.
- All chracters in the sequences $\in \{ \{, \}, (, ), [, ] \}$.

#include <iostream>

```cpp
#include <stack>

#include <string>

using namespace std;


string isBalanced(string s) {

    stack<char> st;

    for (char c : s) {

        if (c == '(' || c == '[' || c == '{') {

            st.push(c); // Push opening brackets onto the stack

        } else {

            if (st.empty()) return "NO"; // No matching opening bracket

            char top = st.top();

            if ((c == ')' && top == '(') ||

                (c == ']' && top == '[') ||

                (c == '}' && top == '{')) {

                st.pop(); // Matched pair, pop the top

            } else {

                return "NO"; // Mismatched pair

            }

        }

    }

    return st.empty() ? "YES" : "NO"; // Balanced if stack is empty

}
```

```cpp
int main() {

    int n;

    cin >> n;

    while (n--) {

        string s;

        cin >> s;

        cout << isBalanced(s) << endl;

    }

    return 0;

}
```
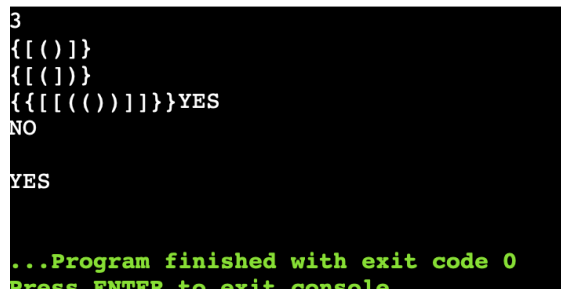
```
3
{[()]}
{[(])}
{{[[(())]]}}YES
NO

YES


...Program finished with exit code 0
Press ENTER to exit console.
```

**4 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.**

**The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step:**

If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue.
Otherwise, they will leave it and go to the queue's end.
This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the ith sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the jth

student in the initial queue (j = 0 is the front of the queue). Return the number of students that are unable to eat.

**Example 1:**

**Input:** students = [1,1,0,0], sandwiches = [0,1,0,1]
**Output:** 0
**Explanation:**
- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1].
- Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0].
- Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,1].
- Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1].
- Front student takes the top sandwich and leaves the line making students = [] and sandwiches = [].
Hence all students are able to eat.

```cpp
#include <iostream>

#include <queue>

#include <vector>

using namespace std;


int countStudents(vector<int>& students, vector<int>& sandwiches) {

    queue<int> q; // Queue for students

    for (int student : students) q.push(student);


    int count = 0; // To track rotations without a successful match

    int i = 0;    // Index for the sandwich stack
```

```cpp
    while (!q.empty() && count < q.size()) {

        if (q.front() == sandwiches[i]) {

            q.pop(); // Student takes the sandwich

            i++;    // Move to the next sandwich

            count = 0; // Reset the counter

        } else {

            q.push(q.front()); // Move the student to the end of the queue

            q.pop();

            count++; // Increment the counter for unmatched attempts

        }

    }


    return q.size(); // Remaining students unable to eat

}


int main() {

    vector<int> students = {1, 1, 0, 0};

    vector<int> sandwiches = {0, 1, 0, 1};


    cout << countStudents(students, sandwiches) << endl; // Output: 0

    return 0;

}
```

**5 We are given an array asteroids of integers representing asteroids in a row.**

**For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.**

**Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.**

**Example 1:**

**Input:** asteroids = [5,10,-5]

**Output:** [5,10]

**Explanation:** The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

**Example 2:**

**Input:** asteroids = [8,-8]

**Output: []**

**Explanation:** The 8 and -8 collide exploding each other.

**Example 3:**

**Input:** asteroids = [10,2,-5]

**Output:** [10]

#include <iostream>

#include <vector>

#include <stack>

```cpp
using namespace std;

vector<int> asteroidCollision(vector<int>& asteroids) {
    stack<int> st; // Stack to simulate asteroid collisions

    for (int asteroid : asteroids) {
        bool destroyed = false;

        while (!st.empty() && asteroid < 0 && st.top() > 0) {
            if (abs(asteroid) == st.top()) {
                st.pop(); // Both asteroids explode
                destroyed = true;
                break;
            } else if (abs(asteroid) > st.top()) {
                st.pop(); // Top asteroid is smaller and explodes
            } else {
                destroyed = true; // Current asteroid is smaller and explodes
                break;
            }
        }

        if (!destroyed) st.push(asteroid); // Push the current asteroid if not destroyed
    }
```

```cpp
    vector<int> result(st.size());

    for (int i = st.size() - 1; i >= 0; i--) {

        result[i] = st.top(); // Transfer stack to result in reverse order

        st.pop();

    }


    return result;

}


int main() {

    vector<int> asteroids1 = {5, 10, -5};

    vector<int> asteroids2 = {8, -8};

    vector<int> asteroids3 = {10, 2, -5};


    vector<int> result1 = asteroidCollision(asteroids1);

    vector<int> result2 = asteroidCollision(asteroids2);

    vector<int> result3 = asteroidCollision(asteroids3);


    for (int x : result1) cout << x << " "; // Output: 5 10

    cout << endl;

    for (int x : result2) cout << x << " "; // Output: (empty)

    cout << endl;
```
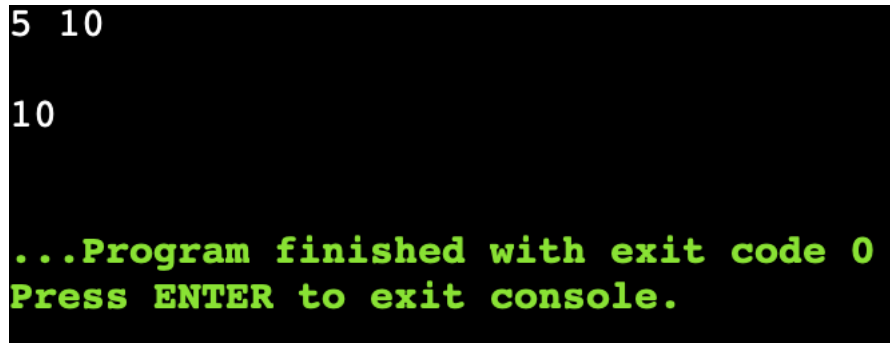
```cpp
    for (int x : result3) cout << x << " "; // Output: 10

    cout << endl;



    return 0;

}
```

```
5 10

10


...Program finished with exit code 0
Press ENTER to exit console.
```

**9 Given an integer array nums, handle multiple queries of the following type:**

**Calculate the sum of the elements of nums between indices left and right inclusive where left <= right.**
**Implement the NumArray class:**

**NumArray(int[] nums) Initializes the object with the integer array nums.**
**int sumRange(int left, int right) Returns the sum of the elements of nums between indices left and right inclusive (i.e. nums[left] + nums[left + 1] + ... + nums[right]).**
**Example 1:**
**Input :["NumArray", "sumRange", "sumRange", "sumRange"]**
**[[[-2, 0, 3, -5, 2, -1]], [0, 2], [2, 5], [0, 5]]**
**Output:[null, 1, -1, -3]**

**Explanation**
NumArray numArray = new NumArray([-2, 0, 3, -5, 2, -1]);
numArray.sumRange(0, 2); // return (-2) + 0 + 3 = 1
numArray.sumRange(2, 5); // return 3 + (-5) + 2 + (-1) = -1
numArray.sumRange(0, 5); // return (-2) + 0 + 3 + (-5) + 2 + (-1) = -3

**Constraints:**

- $1 <= nums.length <= 104$
- $-105 <= nums[i] <= 105$
- $0 <= left <= right < nums.length$
- At most 104 calls will be made to sumRange.

```cpp
#include <iostream>

#include <vector>

using namespace std;


class NumArray {

    vector<int> prefixSum;


public:

    NumArray(vector<int>& nums) {

        int n = nums.size();

        prefixSum.resize(n + 1, 0); // Prefix sum array

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

            prefixSum[i + 1] = prefixSum[i] + nums[i];

        }

    }


    int sumRange(int left, int right) {

        return prefixSum[right + 1] - prefixSum[left];

    }
};


int main() {

    vector<int> nums = {-2, 0, 3, -5, 2, -1};
```

```
    NumArray numArray(nums);



    cout << numArray.sumRange(0, 2) << endl; // Output: 1

    cout << numArray.sumRange(2, 5) << endl; // Output: -1

    cout << numArray.sumRange(0, 5) << endl; // Output: -3



    return 0;

}
```
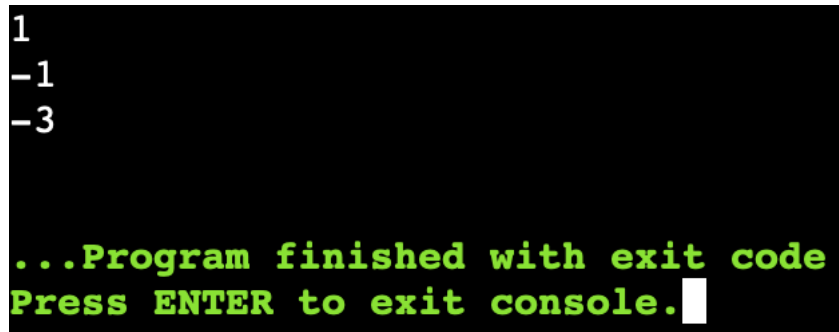
```
1
-1
-3


...Program finished with exit code
Press ENTER to exit console.
```

**10 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]

 **Constraints:**

- 1 <= nums.length <= 104
- -109 <= nums[i] <= 109

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

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

    int n = nums.size();

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

    stack<int> s; // Monotonic stack


    // First pass (left to right)

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

        while (!s.empty() && nums[s.top()] < nums[i]) {

            result[s.top()] = nums[i];

            s.pop();

        }

        s.push(i);

    }
```

```cpp
    // Second pass (right to left for circular)

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

        while (!s.empty() && nums[s.top()] < nums[i]) {

            result[s.top()] = nums[i];

            s.pop();

        }

    }


    return result;

}


int main() {

    vector<int> nums = {1, 2, 1};

    vector<int> result = nextGreaterElements(nums);


    for (int r : result) {

        cout << r << " ";

    }

    cout << endl;


    return 0;

}
```

```
2 -1 2


...Program finished with exit code 0
Press ENTER to exit console.
```