

## DOMAIN WINTER WINNING CAMP

Student Name: Unnati Srivastava

UID: 22BCS13475

Branch: BE-CSE

Section/Group: TPP\_FL\_603-A

### DAY 4:

**QUES 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.

### **Solution:**

```
#include <iostream>
#include <stack>
using namespace std;
class MinStack {
    stack<int> mainStack;
    stack<int> minStack;
public:
    MinStack() {}
    void push(int val) {
        mainStack.push(val);
        if (minStack.empty() || val <= minStack.top()) {
            minStack.push(val);
        }
    }
};
```

```
}  
  
void pop() {  
    if (mainStack.top() == minStack.top()) {  
        minStack.pop();  
    }  
    mainStack.pop();  
}  
  
int top() {  
    return mainStack.top();  
}  
  
int getMin() {  
    return minStack.top();  
}  
};  
  
int main() {  
    MinStack minStack;  
    minStack.push(-2);  
    minStack.push(0);  
    minStack.push(-3);  
    cout << minStack.getMin() << endl; // Output: -3  
    minStack.pop();  
    cout << minStack.top() << endl; // Output: 0  
    cout << minStack.getMin() << endl; // Output: -2  
    MinStack anotherStack;  
    anotherStack.push(5);  
    anotherStack.push(3);  
    anotherStack.push(7);  
    anotherStack.push(3);  
    cout << anotherStack.getMin() << endl; // Output: 3
```

```
anotherStack.pop();  
  
cout << anotherStack.getMin() << endl; // Output: 3  
  
cout << anotherStack.top() << endl; // Output: 7  
  
cout << anotherStack.getMin() << endl; // Output: 3  
  
return 0;  
  
}
```



```
-3  
0  
-2  
3  
3  
7  
3
```

**QUES 2:** 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.

**Solution:**

```
#include <iostream>  
  
#include <queue>  
  
#include <vector>  
  
using namespace std;  
  
int countStudents(vector<int>& students, vector<int>& sandwiches) {  
    queue<int> studentQueue;  
    for (int student : students) {  
        studentQueue.push(student);  
    }  
    int index = 0, rotations = 0;  
    while (!studentQueue.empty() && rotations < studentQueue.size()) {
```

```
        if (studentQueue.front() == sandwiches[index]) {
            studentQueue.pop();
            index++;
            rotations = 0;
        } else {
            studentQueue.push(studentQueue.front());
            studentQueue.pop();
            rotations++;
        }
    }
    return studentQueue.size();
}

int main() {
    vector<int> students1 = {1, 1, 0, 0};
    vector<int> sandwiches1 = {0, 1, 0, 1};
    cout << countStudents(students1, sandwiches1) << endl; // Output: 0
    vector<int> students2 = {1, 1, 1, 0, 0, 1};
    vector<int> sandwiches2 = {1, 0, 0, 0, 1, 1};
    cout << countStudents(students2, sandwiches2) << endl; // Output: 3
    return 0;
}
```



```
0
3
```

**QUES 3:** 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.

**Solution:**

```
#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> s;
    for (int i = 0; i < 2 * n; ++i) {
        while (!s.empty() && nums[s.top()] < nums[i % n]) {
            result[s.top()] = nums[i % n];
            s.pop();
        }
        if (i < n) {
            s.push(i);
        }
    }
    return result;
}

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

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

    cout << endl;
```

```
    return 0;  
}
```



2 -1 2

**QUES 4:** 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.

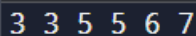
Return the max sliding window.

**Solution:**

```
#include <iostream>  
#include <vector>  
#include <deque>  
using namespace std;  
vector<int> maxSlidingWindow(vector<int>& nums, int k) {  
    deque<int> dq;  
    vector<int> result;  
    for (int i = 0; i < nums.size(); ++i) {  
        if (!dq.empty() && dq.front() == i - k) {  
            dq.pop_front();  
        }  
        while (!dq.empty() && nums[dq.back()] < nums[i]) {  
            dq.pop_back();  
        }  
        dq.push_back(i);  
        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 num : result) {
            cout << num << " ";
        }
        cout << endl;
        return 0;
    }
```



**QUES 5:** There are a number of plants in a garden. Each of the plants has been treated with some amount of pesticide. After each day, if any plant has more pesticide than the plant on its left, being weaker than the left one, it dies.

You are given the initial values of the pesticide in each of the plants. Determine the number of days after which no plant dies, i.e. the time after which there is no plant with more pesticide content than the plant to its left.

**Solution:**

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

int poisonousPlants(vector<int>& p) {
    vector<int> days(p.size(), 0);
    stack<int> s;
```

```
for (int i = 0; i < p.size(); ++i) {  
    int maxDays = 0;  
    while (!s.empty() && p[s.top()] >= p[i]) {  
        maxDays = max(maxDays, days[s.top()]);  
        s.pop();  
    }  
    if (!s.empty()) {  
        days[i] = maxDays + 1;  
    }  
    s.push(i);  
}  
return *max_element(days.begin(), days.end());  
}  
int main() {  
    vector<int> p = {3, 6, 2, 7, 5};  
    cout << poisonousPlants(p) << endl; // Output: 2  
  
    return 0;  
}
```