DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Gurnoor Oberoi **UID: 22BCS15716**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

➢ DAY-4 [23-12-2024]

1. Min Stack *(Very Easy)*

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.

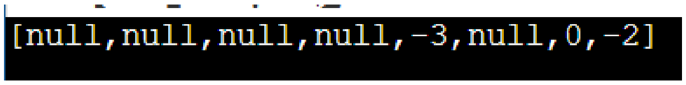
You must implement a solution with O(1) time complexity for each function.

Implementation/Code:

```
#include <iostream>
#include <stack>
#include <vector>
#include <string>
using namespace std;
class MinStack {
private:
    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.empty()) return;
        if (mainStack.top() == minStack.top()) {
            minStack.pop();
        }
        mainStack.pop();
    }
    int top() {
        return mainStack.top();
    }
    int getMin() {
        return minStack.top();
    }
};

int main() {
    vector<string> operations = {"MinStack", "push", "push", "push", "getMin", "pop",
"top", "getMin"};
    vector<vector<int>> values = {{}, {-2}, {0}, {-3}, {}, {}, {}, {}};
    vector<int> output;
    MinStack* obj = nullptr;
    for (size_t i = 0; i < operations.size(); i++) {
        if (operations[i] == "MinStack") {
            obj = new MinStack();
            output.push_back(-1);
        } else if (operations[i] == "push") {
            obj->push(values[i][0]);
            output.push_back(-1);
        } else if (operations[i] == "pop") {
            obj->pop();
            output.push_back(-1);
        } else if (operations[i] == "top") {
            output.push_back(obj->top());
        } else if (operations[i] == "getMin") {
            output.push_back(obj->getMin());
        }
    }
    cout << "[";
    for (size_t i = 0; i < output.size(); i++) {
        if (i > 0) cout << ",";
        if (output[i] == -1)
            cout << "null";
        else
            cout << output[i];
    }
    cout << "]" << endl;
    delete obj;
    return 0;
}
```

Output:



```
[null,null,null,null,-3,null,0,-2]
```

2. Number of Students Unable to Eat Lunch *(Easy)*

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.

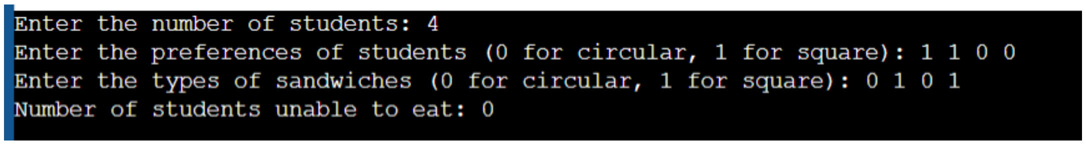
Implementation/Code:

```
#include <iostream>
```

```
#include <vector>
using namespace std;
int countStudents(vector<int>& students, vector<int>& sandwiches) {
    int countZero = 0, countOne = 0;
    for (int student : students) {
        if (student == 0) countZero++;
        else countOne++;
    }
    for (int sandwich : sandwiches) {
        if (sandwich == 0) {
            if (countZero > 0) countZero--;
            else return countZero + countOne;
        } else {
            if (countOne > 0) countOne--;
            else return countZero + countOne;
        }
    }
    return 0;
}

int main() {
    int n;
    cout << "Enter the number of students: ";
    cin >> n;
    vector<int> students(n), sandwiches(n);
    cout << "Enter the preferences of students (0 for circular, 1 for square): ";
    for (int i = 0; i < n; i++) {
        cin >> students[i];
    }
    cout << "Enter the types of sandwiches (0 for circular, 1 for square): ";
    for (int i = 0; i < n; i++) {
        cin >> sandwiches[i];
    }
    int result = countStudents(students, sandwiches);
    cout << "Number of students unable to eat: " << result << endl;
    return 0;
}
```

Output:



```
Enter the number of students: 4
Enter the preferences of students (0 for circular, 1 for square): 1 1 0 0
Enter the types of sandwiches (0 for circular, 1 for square): 0 1 0 1
Number of students unable to eat: 0
```

3. Next Greater Element II *(Medium)*

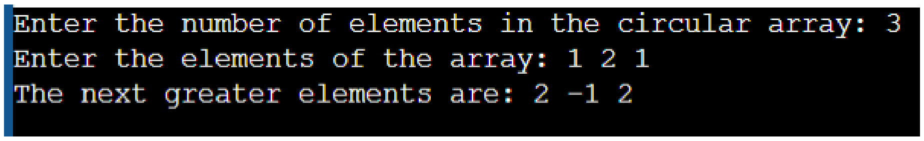
Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return the next greater number for every element in nums.
The next greater number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number.

Implementation/Code:

```
#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++) {
        int current = nums[i % n];
        while (!st.empty() && nums[st.top()] < current) {
            result[st.top()] = current;
            st.pop();
        }
        if (i < n) {
            st.push(i);
        }
    }
    return result;
}

int main() {
    int n;
    cout << "Enter the number of elements in the circular array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
    vector<int> result = nextGreaterElements(nums);
    cout << "The next greater elements are: ";
    for (int i : result) {
        cout << i << " ";
    }
    cout << endl;
    return 0;
}
```

Output:



```
Enter the number of elements in the circular array: 3
Enter the elements of the array: 1 2 1
The next greater elements are: 2 -1 2
```

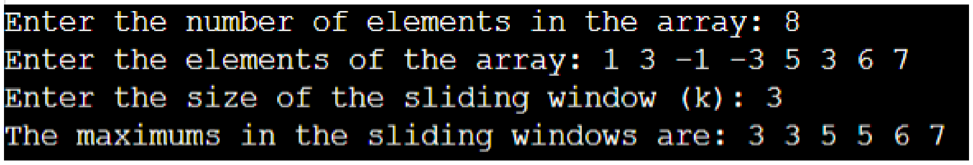
4. Sliding Window *(Hard)*

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.

Implementation/Code:

```
#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++) {
        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() {
    int n, k;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
    cout << "Enter the size of the sliding window (k): ";
    cin >> k;
    vector<int> result = maxSlidingWindow(nums, k);
    cout << "The maximums in the sliding windows are: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

Output:



```
Enter the number of elements in the array: 8
Enter the elements of the array: 1 3 -1 -3 5 3 6 7
Enter the size of the sliding window (k): 3
The maximums in the sliding windows are: 3 3 5 5 6 7
```

5. Poisonous Plants *(Very Hard)*

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.

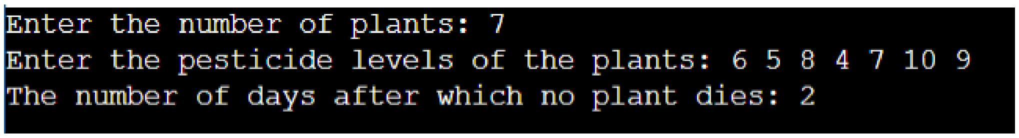
Implementation/Code:

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
int pesticideDays(vector<int>& plants) {
    int n = plants.size();
    vector<int> days(n, 0);
    stack<int> st;
    int maxDays = 0;
    for (int i = 0; i < n; i++) {
        while (!st.empty() && plants[i] > plants[st.top()]) {
            days[i] = max(days[i], days[st.top()] + 1);
            st.pop();
        }
        maxDays = max(maxDays, days[i]);
        st.push(i);
    }

    return maxDays;
}
int main() {
    int n;
    cout << "Enter the number of plants: ";
    cin >> n;
    vector<int> plants(n);
    cout << "Enter the pesticide levels of the plants: ";
    for (int i = 0; i < n; i++) {
        cin >> plants[i];
    }
    int result = pesticideDays(plants);
    cout << "The number of days after which no plant dies: " << result << endl;

    return 0;
}
```

Output:



```
Enter the number of plants: 7
Enter the pesticide levels of the plants: 6 5 8 4 7 10 9
The number of days after which no plant dies: 2
```