# DOMAIN WINTER CAMP

# DAY - 4

**Student Name:** UDIT SHARMA **UID:** 22BCS12728

**Branch:** BE-CSE **Section:** 22BCS_FL_IOT-603

## STACK AND QUEUE STANDARD QUESTION

### VERY EASY:

**Q. 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.

**Solution:**

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

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

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() {
    int queries;
    cout << "Enter the number of queries: ";
    cin >> queries;

    MinStack minStack;
    while (queries--) {
        string command;
        cout << "Enter command (push, pop, top, getMin): ";
        cin >> command;

        if (command == "push") {
            int val;
            cout << "Enter value to push: ";
            cin >> val;
            minStack.push(val);
        } else if (command == "pop") {
            minStack.pop();
            cout << "Element popped from the stack.\n";
        } else if (command == "top") {
            cout << "Top element: " << minStack.top() << endl;
        } else if (command == "getMin") {
            cout << "Minimum element: " << minStack.getMin() << endl;
        } else {
            cout << "Invalid command.\n";
        }
```

```
    }

    return 0;
}
```

**Output:**

```
Enter the number of queries: 8
Enter command (push, pop, top, getMin): push
Enter value to push: 4
Enter command (push, pop, top, getMin): push
Enter value to push: -2
Enter command (push, pop, top, getMin): push
Enter value to push: 2
Enter command (push, pop, top, getMin): push
Enter value to push: 1
Enter command (push, pop, top, getMin): top
Top element: 1
Enter command (push, pop, top, getMin): pop
Element popped from the stack.
Enter command (push, pop, top, getMin): getMin
Minimum element: -2
Enter command (push, pop, top, getMin): top
Top element: 2


=== Code Execution Successful ===
```

## EASY LEVEL:-

**Q. 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.

**Solution:**

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

int countStudents(vector<int>& students, vector<int>& sandwiches) {
    queue<int> studentQueue, sandwichStack;
    int count = 0;

    // Fill queues with initial data
    for (int student : students) studentQueue.push(student);
    for (int sandwich : sandwiches) sandwichStack.push(sandwich);

    while (!studentQueue.empty() && count < studentQueue.size()) {
        if (studentQueue.front() == sandwichStack.front()) {
            // Student takes the sandwich
            studentQueue.pop();
            sandwichStack.pop();
            count = 0; // Reset the loop counter
        } else {
            // Student goes to the end of the queue
            studentQueue.push(studentQueue.front());
            studentQueue.pop();
            count++; // Increase loop counter
        }
    }

    return studentQueue.size(); // Remaining students
}

int main() {
    int n;
    cout << "Enter the number of students and sandwiches: ";
    cin >> n;

    vector<int> students(n), sandwiches(n);
    cout << "Enter student preferences (0 or 1): ";
    for (int i = 0; i < n; ++i) cin >> students[i];

    cout << "Enter sandwich types (0 or 1): ";
    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 and sandwiches: 6
Enter student preferences (0 or 1): 1 1 1 0 0 1
Enter sandwich types (0 or 1): 1 0 0 0 1 1
Number of students unable to eat: 3


=== Code Execution Successful ===
```

## MEDIUM:-

**Q. 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;

    // Iterate through the array twice to handle the circular property
    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() {
    int n;
    cout << "Enter the size of 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];

    vector<int> result = nextGreaterElements(nums);
    cout << "Next greater elements: ";
    for (int r : result) cout << r << " ";
    cout << endl;

    return 0;
}
```

**Output:**

```
Enter the size of the array: 5
Enter the elements of the array: 1 2 3 4 3
Next greater elements: 2 3 4 -1 4


=== Code Execution Successful ===
```

## HARD

**Q. 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) {
    vector<int> result;
    deque<int> dq; // Stores indices of useful elements in the current window

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

        // Remove smaller elements as they are not useful
        while (!dq.empty() && nums[dq.back()] < nums[i]) {
```

```
            dq.pop_back();
        }

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

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

    return result;
}

int main() {
    int n, k;
    cout << "Enter the size of 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 window size: ";
    cin >> k;

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

    cout << "Maximum sliding window: ";
    for (int r : result) {
        cout << r << " ";
    }
    cout << endl;

    return 0;
}
```

**Output:**

```
Enter the size of the array: 8
Enter the elements of the array: 1 3 -1 -3 5 3 6 7
Enter the window size: 3
Maximum sliding window: 3 3 5 5 6 7


=== Code Execution Successful ===
```

## VERY HARD

**Q. 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) {
    int n = p.size();
    vector<int> days(n, 0); // Days required for each plant to die
    stack<int> st;          // Stack to track plant indices

    int maxDays = 0;
    for (int i = 0; i < n; ++i) {
        while (!st.empty() && p[st.top()] >= p[i]) {
            st.pop(); // Remove plants with greater or equal pesticide level
        }

        if (!st.empty()) {
            days[i] = days[st.top()] + 1; // Calculate days to die
        }

        maxDays = max(maxDays, days[i]); // Track maximum days
        st.push(i); // Push current plant index onto the stack
    }

    return maxDays;
```

```
}

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

    vector<int> p(n);
    cout << "Enter the pesticide levels of plants: ";
    for (int i = 0; i < n; ++i) {
        cin >> p[i];
    }

    int result = poisonousPlants(p);
    cout << "Number of days until no plants die: " << result << endl;
    return 0;
}
```

**Output:**

```
Enter the number of plants: 5
Enter the pesticide levels of plants: 3 6 2 7 5
Number of days until no plants die: 2


=== Code Execution Successful ===
```