# DOMAIN WINTER CAMP

(Department of Computer Science and Engineering)

**Name: Ankit Vashisth  UID: 22BCS13378  Section: KPIT 901-B**

## DAY-4

**(Easy)**

**Q1 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.**

**Example 1:**

**Input**

**["MinStack","push","push","push","getMin","pop","top","getMin"]**

**[[],[-2],[0],[-3],[],[],[],[]]**

**Program code:**

```
#include <iostream>
```

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

class MinStack {
private:
    stack<int> mainStack;   // Stack to hold all elements
    stack<int> minStack;    // Stack to keep track of minimum
elements

public:
    MinStack() {}

    void push(int val) {
        mainStack.push(val);
        if (minStack.empty() || val <= minStack.top()) {
            minStack.push(val);
        }
    }

    void pop() {
        if (!mainStack.empty() && 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>> inputs = {{}, {-2}, {0}, {-3}, {}, {},
{}, {}};
    vector<string> output;

    MinStack* minStack = nullptr;

    for (size_t i = 0; i < operations.size(); ++i) {
        if (operations[i] == "MinStack") {
            minStack = new MinStack();
            output.push_back("null");
        } else if (operations[i] == "push") {
            minStack->push(inputs[i][0]);
            output.push_back("null");
        } else if (operations[i] == "pop") {
            minStack->pop();
            output.push_back("null");
        } else if (operations[i] == "top") {
            output.push_back(to_string(minStack->top()));
        } else if (operations[i] == "getMin") {
            output.push_back(to_string(minStack->getMin()));
        }
    }

    // Print the output
    cout << "[";
    for (size_t i = 0; i < output.size(); ++i) {
        cout << output[i];
        if (i < output.size() - 1) cout << ",";
    }
    cout << "]" << endl;

    return 0;
}
```

**Output:**

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ cpp_basic.cpp -
o cpp_basic && ./cpp_basic
[null,null,null,null,-3,null,0,-2]
```

**Q 2 (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.**

**Example 1:**

**Input: nums = [1,2,1]**

**Output: [2,-1,2]**

**Program 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); // Initialize the result array with -
1
   stack<int> st;            // Monotonic stack to store indices

   // Traverse the array twice to handle the circular nature
   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 << "[";
    for (size_t i = 0; i < result.size(); i++) {
        cout << result[i];
        if (i < result.size() - 1) cout << ",";
    }
    cout << "]" << endl;

    return 0;
}
```

**Output:**

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ oneday.cpp -o o
neday && ./oneday
[2,-1,2]
```

**Ques 3. Given a queue, write a recursive function to reverse it.**

**Standard operations allowed :**

**enqueue(x) : Add an item x to rear of queue.**
**dequeue() : Remove an item from front of queue.**
**empty() : Checks if a queue is empty or not.**

**Examples 1:**

**Input : Q = [5, 24, 9, 6, 8, 4, 1, 8, 3, 6]**

**Output : Q = [6, 3, 8, 1, 4, 8, 6, 9, 24, 5]**

**Program Code:**

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

// Function to reverse the queue recursively
void reverseQueue(queue<int>& q) {
    // Base case: if the queue is empty, return
    if (q.empty()) {
        return;
    }

    // Step 1: Remove the front element of the queue
    int front = q.front();
    q.pop();
```

```
    // Step 2: Recursively reverse the remaining queue
    reverseQueue(q);

    // Step 3: Add the removed element to the back of the queue
    q.push(front);
}

int main() {
    // Input queue
    queue<int> Q;
    Q.push(5);
    Q.push(24);
    Q.push(9);
    Q.push(6);
    Q.push(8);
    Q.push(4);
    Q.push(1);
    Q.push(8);
    Q.push(3);
    Q.push(6);

    // Print original queue
    cout << "Original Queue: ";
    queue<int> tempQ = Q; // Temporary queue to preserve original for printing
    while (!tempQ.empty()) {
        cout << tempQ.front() << " ";
        tempQ.pop();
    }
    cout << endl;

    // Reverse the queue
    reverseQueue(Q);

    // Print reversed queue
    cout << "Reversed Queue: ";
    while (!Q.empty()) {
        cout << Q.front() << " ";
        Q.pop();
    }
    cout << endl;
```

```
    return 0;
}
```

**Output:**

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ array.cpp -o ar
ray && ./array
Original Queue: 5 24 9 6 8 4 1 8 3 6
Reversed Queue: 6 3 8 1 4 8 6 9 24 5
```

**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.**

**Example 1:**

**Input: nums = [1,3,-1,-3,5,3,6,7], k = 3**

**Output: [3,3,5,5,6,7]**

**Explanation:**

| Window position | Max |
|---|---|
| [1 3 -1] -3 5 3 6 7 | 3 |
| 1 [3 -1 -3] 5 3 6 7 | 3 |
| 1 3 [-1 -3 5] 3 6 7 | 5 |
| 1 3 -1 [-3 5 3] 6 7 | 5 |
| 1 3 -1 -3 [5 3 6] 7 | 6 |
| 1 3 -1 -3 5 [3 6 7] | 7 |

**Program Code:**

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

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

```
    deque<int> dq; // Stores indices of array elements
    vector<int> result;

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

        // Remove elements from deque that are smaller than the current element
        while (!dq.empty() && nums[dq.back()] < nums[i]) {
            dq.pop_back();
        }

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

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

    return result;
}

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

    // Print the result
    for (int x : result) {
        cout << x << " ";
    }
    cout << endl;

    return 0;
}
```

**Output**:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ tw2.cpp -o tw2
&& ./tw2
3 3 5 5 6 7
```

**Ques 5 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.**

**Implement the DinnerPlates class:**

**DinnerPlates(int capacity) Initializes the object with the maximum capacity of the stacks capacity.**

**void push(int val) Pushes the given integer val into the leftmost stack with a size less than capacity.**

**int pop() Returns the value at the top of the rightmost non-empty stack and removes it from that stack, and returns -1 if all the stacks are empty.**

**int popAtStack(int index) Returns the value at the top of the stack with the given index index and removes it from that stack or returns -1 if the stack with that given index is empty.**

**Program Code:**

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

class DinnerPlates {
private:
    int capacity;
    vector<stack<int>> stacks;
    int leftMostStack;

public:
    // Constructor to initialize the DinnerPlates object
    DinnerPlates(int capacity) {
```

```
        this->capacity = capacity;
        this->leftMostStack = 0;
        cout << "null" << endl; // Output null for constructor
    }

    // Pushes the value into the leftmost stack with size less than capacity
    void push(int val) {
        while (leftMostStack < stacks.size() && stacks[leftMostStack].size() ==
capacity) {
            leftMostStack++;
        }

        if (leftMostStack == stacks.size()) {
            stacks.push_back(stack<int>());
        }

        stacks[leftMostStack].push(val);
        cout << "null" << endl; // Output null for push
    }

    // Pops the top value from the rightmost non-empty stack and removes it
    int pop() {
        if (stacks.empty()) {
            return -1;
        }

        while (!stacks.empty() && stacks.back().empty()) {
            stacks.pop_back();
        }

        if (stacks.empty()) {
            return -1;
        }

        int val = stacks.back().top();
        stacks.back().pop();
        return val;
    }

    // Pops the top value from the stack at the given index and removes it
    int popAtStack(int index) {
        if (index >= stacks.size() || stacks[index].empty()) {
```

```
            return -1;
        }

        int val = stacks[index].top();
        stacks[index].pop();
        return val;
    }
};

int main() {
    DinnerPlates dp(2);  // Constructor call, should output 'null'

    dp.push(1);  // Should output 'null'
    dp.push(2);  // Should output 'null'
    dp.push(3);  // Should output 'null'
    dp.push(4);  // Should output 'null'
    dp.push(5);  // Should output 'null'

    cout << dp.popAtStack(0) << endl; // 2
    dp.push(20);  // Should output 'null'
    dp.push(21);  // Should output 'null'

    cout << dp.popAtStack(0) << endl; // 20
    cout << dp.popAtStack(2) << endl; // 21
    cout << dp.pop() << endl;         // 5
    cout << dp.pop() << endl;         // 4
    cout << dp.pop() << endl;         // 3
    cout << dp.pop() << endl;         // 1
    cout << dp.pop() << endl;         // -1

    return 0;
}
```

**Output:**

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ two.cpp -o two
&& ./two
null
null
null
null
null
null
2
null
null
1
20
21
5
4
3
-1
```