



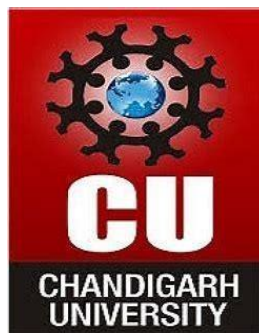
**CHANDIGARH
UNIVERSITY**
Discover. Learn. Empower.

**NAAC
GRADE A+**
Accredited University

UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE)



WINTER DOMAIN CAMP

Date : 24/12/2024

Submitted to:

Faculty name: Er.Rajni Devi

Submitted by:

Name: Akshi Datta

UID: 22BCS15369

Section: IOT-620

DAY 4(24/12/24)

PROBLEM 1

Objective: Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Code:

```
#include <stack>
#include <iostream>
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()) {
            if (mainStack.top() == minStack.top()) {
                minStack.pop();
            }
            mainStack.pop();
        }
    }
    int top() {
        if (!mainStack.empty()) {
            return mainStack.top();
        }
        throw runtime_error("Stack is empty.");
    }
    int getMin() {
        if (!minStack.empty()) {
            return minStack.top();
        }
        throw runtime_error("Stack is empty.");
    }
};


int main() {
    try {
        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
} catch (const exception &e) {
cerr << e.what() << endl;
}
return 0;
}

```

Output:



```

Output
-3
0
-2

```

PROBLEM 2

Objective: 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 *i*th sandwich in the stack (*i* = 0 is the top of the stack) and `students[j]` is the preference of the *j*th student in the initial queue (*j* = 0 is the front of the queue). Return the number of students that are unable to eat.

Code:

```

#include <iostream>
#include <queue>
#include <vector>
using namespace std;
int countStudents(vector<int>& students, vector<int>& sandwiches) {
queue<int> studentQueue;
queue<int> sandwichStack;
for (int s : students) {
studentQueue.push(s);
}
for (int s : sandwiches) {
sandwichStack.push(s);
}
}

```

```

}
int loopCount = 0; // Counter to detect a loop
int totalStudents = students.size();
while (!studentQueue.empty() && !sandwichStack.empty()) {
    if (studentQueue.front() == sandwichStack.front()) {
        studentQueue.pop();
        sandwichStack.pop();
        loopCount = 0; // Reset loop counter
    } else {
        studentQueue.push(studentQueue.front());
        studentQueue.pop();
        loopCount++;
    }
    if (loopCount == totalStudents) {
        break;
    }
}

// Return the number of students left in the queue
return studentQueue.size();
}

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

    return 0;
}

```

Output:

```

Output
Result: 0
Result: 3

```

PROBLEM 3

Objective: 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..

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 currentIndex = i % n; // Circular index
        while (!st.empty() && nums[st.top()] < nums[currentIndex]) {
            result[st.top()] = nums[currentIndex];
            st.pop();
        }
        if (i < n) {
            st.push(currentIndex);
        }
    }

    return result;
}

int main() {
    vector<int> nums1 = {1, 2, 1};
    vector<int> result1 = nextGreaterElements(nums1);
    for (int val : result1) {
        cout << val << " ";
    }
    cout << endl; // Output: 2 -1 2

    vector<int> nums2 = {1, 2, 3, 4, 3};
    vector<int> result2 = nextGreaterElements(nums2);
    for (int val : result2) {
        cout << val << " ";
    }
    cout << endl; // Output: 2 3 4 -1 4

    return 0;
}
```

Output:

Output

```
2 -1 2
2 3 4 -1 4
```

PROBLEM 4

Objective: 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.

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() {
    vector<int> nums1 = {1, 3, -1, -3, 5, 3, 6, 7};
    int k1 = 3;
    vector<int> result1 = maxSlidingWindow(nums1, k1);
    for (int val : result1) {
        cout << val << " ";
    }
    cout << endl; // Output: 3 3 5 5 6 7

    vector<int> nums2 = {1};
    int k2 = 1;
```

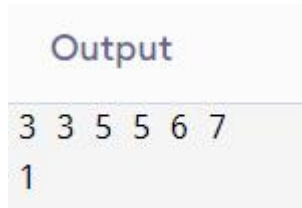
```

vector<int> result2 = maxSlidingWindow(nums2, k2);
for (int val : result2) {
    cout << val << " ";
}
cout << endl; // Output: 1

return 0;
}

```

Output:



```

Output
3 3 5 5 6 7
1

```

PROBLEM 5

Objective: 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.

Code:

```

#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
using namespace std;
int poisonousPlants(vector<int>& p) {
    stack<pair<int, int>> s;
    int maxDays = 0;
    for (int pesticide : p) {
        int days = 0;
        while (!s.empty() && pesticide > s.top().first) {
            days = max(days, s.top().second);
            s.pop();
        }
        if (!s.empty()) {
            days += 1;
        } else {
            days = 0;
        }
        s.push({pesticide, days});
        maxDays = max(maxDays, days);
    }
}

```

```

    }

    return maxDays;
}

int main() {
    vector<int> p1 = {3, 6, 2, 7, 5};
    cout << "Example 1: " << poisonousPlants(p1) << endl; // Output: 2
    // Example 2
    vector<int> p2 = {6, 5, 8, 4, 7, 10, 9};
    cout << "Example 2: " << poisonousPlants(p2) << endl; // Output: 2
    return 0;
}

```

Output:

```

Output
Example 1: 1
Example 2: 2

```

PROBLEM 6

Objective: 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.

Code:

```

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

bool isBalanced(string s) {
    stack<char> st;
    for (char c : s) {
        if (c == '(' || c == '{' || c == '[') {
            st.push(c);
        } else {

```



```

        if (st.empty()) return false;
        if ((c == ')' && st.top() != '(') ||
            (c == '}' && st.top() != '{') ||
            (c == ']' && st.top() != '[')) {
            return false;
        }
        st.pop();
    }
}
return st.empty();
}

int main() {
    int n;
    cout << "Enter number of bracket strings to check: ";
    cin >> n;

    for (int i = 0; i < n; ++i) {
        string s;
        cout << "Enter string: ";
        cin >> s;
        cout << (isBalanced(s) ? "YES" : "NO") << endl;
    }

    return 0;
}

```

Output:

Output

```

Enter number of bracket strings to check: 2
Enter string: ({}))
YES
Enter string: {{[]}]
NO

```

PROBLEM 7

Objective: Evaluate reversed polish notation

Code:

```

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

```

```

#include <string>
using namespace std;

int evalRPN(vector<string>& tokens) {
    stack<int> st;
    for (string& token : tokens) {
        if (token == "+" || token == "-" || token == "*" || token == "/") {
            int b = st.top();
            st.pop();
            int a = st.top();
            st.pop();
            if (token == "+") {
                st.push(a + b);
            } else if (token == "-") {
                st.push(a - b);
            } else if (token == "*") {
                st.push(a * b);
            } else if (token == "/") {
                st.push(a / b);
            }
        } else {
            st.push(stoi(token));
        }
    }
    return st.top();
}

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

    vector<string> tokens(n);
    cout << "Enter the tokens: ";
    for (int i = 0; i < n; ++i) {
        cin >> tokens[i];
    }
    cout << "Result: " << evalRPN(tokens) << endl;
    return 0;
}

```

Output:

Output

```

Enter number of tokens: 5
Enter the tokens: 4 13 5 / +
Result: 6

```

PROBLEM 8

Objective: 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.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int poisonousPlants(vector<int>& p) {
    vector<int> days(p.size(), 0);
    vector<int> stack;
    int maxDays = 0;
    for (int i = 0; i < p.size(); ++i) {
        int day = 0;
        while (!stack.empty() && p[stack.back()] >= p[i]) {
            day = max(day, days[stack.back()]);
            stack.pop_back();
        }
        if (!stack.empty()) {
            days[i] = day + 1;
        }
        stack.push_back(i);
        maxDays = max(maxDays, days[i]);
    }
    return maxDays;
}
int main() {
    int n;
    cout << "Enter number of plants: ";
    cin >> n;
    vector<int> p(n);
    cout << "Enter pesticide levels: ";
    for (int i = 0; i < n; ++i) {
        cin >> p[i];
    }
    cout << "Days until no plants die: " << poisonousPlants(p) << endl;
    return 0;
}
```

Output:

Output

```
Enter number of plants: 10
Enter pesticide levels: 5 1 3 8 9 20 5 17 26 50
Days until no plants die: 2
```

PROBLEM 9

Objective: Implement Queue Using Stacks

Code:

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

class MyQueue {
    stack<int> inStack, outStack;

    void transfer() {
        while (!inStack.empty()) {
            outStack.push(inStack.top());
            inStack.pop();
        }
    }

public:
    void push(int x) {
        inStack.push(x);
    }

    int pop() {
        if (outStack.empty()) transfer();
        int val = outStack.top();
        outStack.pop();
        return val;
    }

    int peek() {
        if (outStack.empty()) transfer();
        return outStack.top();
    }

    bool empty() {
        return inStack.empty() && outStack.empty();
    }
};
```

```

    }
};

int main() {
    MyQueue q;
    int n, op, x;
    cout << "Enter number of operations: ";
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cout << "Choose operation (1: push, 2: pop, 3: peek, 4: empty): ";
        cin >> op;
        if (op == 1) {
            cout << "Enter value to push: ";
            cin >> x;
            q.push(x);
        } else if (op == 2) {
            cout << "Popped: " << q.pop() << endl;
        } else if (op == 3) {
            cout << "Front: " << q.peek() << endl;
        } else if (op == 4) {
            cout << "Empty: " << (q.empty() ? "Yes" : "No") << endl;
        }
    }
    return 0;
}

```

Output:

Output

```

Enter number of operations: 5
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 50
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 60
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 90
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 3
Front: 50
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 2
Popped: 50

```

PROBLEM 10

Objective: Reverse a Queue Using Recursion

Code:

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

void reverseQueue(queue<int>& q) {
    if (q.empty()) return;
    int front = q.front();
    q.pop();
    reverseQueue(q);
    q.push(front);
}

int main() {
    int n, val;
    queue<int> q;
    cout << "Enter number of elements in the queue: ";
    cin >> n;
    cout << "Enter elements: ";
    for (int i = 0; i < n; ++i) {
        cin >> val;
        q.push(val);
    }
    reverseQueue(q);
    cout << "Reversed queue: ";
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    return 0;
}
```

Output:

Output

```
Enter number of elements in the queue: 7
Enter elements: 20 30 40 50 60 77 9
Reversed queue: 9 77 60 50 40 30 20
```