DAY 4

Student Name: RAJ SATYAM **UID: 22BCS15256**
Branch: BE-CSE **Section/Group: 620 - A**
Date of Performance:24/12/24

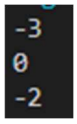
Problem 1

1. **Aim: Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.**
2. **Code:**

```
#include <stack>
#include <limits>
class MinStack {
private:
   std::stack<int> mainStack; // Stack to store the actual elements
   std::stack<int> minStack;   // Stack to store the minimum elements
public:
   MinStack() {
   }
   void push(int val) {
      mainStack.push(val);
      // If minStack is empty or the current value is less than or equal to the
top of minStack
      if (minStack.empty() || val <= minStack.top()) {
         minStack.push(val); // Push the new minimum onto minStack
      }
   }
   void pop() {
      if (mainStack.empty()) return; // Check if the stack is empty
      int topValue = mainStack.top();
      mainStack.pop();
```

```
        if (topValue == minStack.top()) {
            minStack.pop();
        }
    }
    int top() {
        return mainStack.top(); // Return the top element of the main stack
    }
    int getMin() {
        return minStack.top(); // Return the top element of the min stack
    }
};
int main() {
    MinStack minStack;
    minStack.push(-2);
    minStack.push(0);
    minStack.push(-3);
    std::cout << minStack.getMin() << std::endl; // return -3
    minStack.pop();
    std::cout << minStack.top() << std::endl;    // return 0
    std::cout << minStack.getMin() << std::endl; // return -2
    return 0;
}
```

3. **Output:**



Problem 2

1. **Aim: 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**

2. Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
int countStudentsUnableToEat(vector<int>& students, vector<int>&
sandwiches) {
    queue<int> studentQueue;
    for (int student : students) {
        studentQueue.push(student);
    }
    int sandwichIndex = 0;
    int n = sandwiches.size();
    while (!studentQueue.empty() && sandwichIndex < n) {
        int currentStudent = studentQueue.front();
        studentQueue.pop();
        if (currentStudent == sandwiches[sandwichIndex]) {
            sandwichIndex++;
        } else {
            studentQueue.push(currentStudent);
        }
        if (studentQueue.size() == students.size()) {
            break;
        }
    }
    return studentQueue.size();
}
int main() {
    vector<int> students = {1, 1, 0, 0};
    vector<int> sandwiches = {0, 1, 0, 1};
    int result = countStudentsUnableToEat(students, sandwiches);
    cout << "Number of students unable to eat: " << result << endl;
    return 0;
```

}

3. **Output:**

```
Number of students unable to eat: 4
```

Problem 3

1. **Aim: 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.**

2. **Code:**

```
#include <iostream>
#include <vector>
#include <stack>
std::vector<int> nextGreaterElements(std::vector<int>& nums) {
    int n = nums.size();
    std::vector<int> result(n, -1);
    std::stack<int> s;
    for (int i = 0; i < 2 * n; ++i) {
        int currentIndex = i % n;
        while (!s.empty() && nums[currentIndex] > nums[s.top()]) {
            int index = s.top();
            s.pop();
            result[index] = nums[currentIndex];
        }
        if (i < n) {
            s.push(currentIndex);
        }
    }
    return result;
}
```

```
int main() {
    std::vector<int> nums = {1, 2, 1};
    std::vector<int> result = nextGreaterElements(nums);
    for (int num : result) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

3. **Output:**

```
2 -1 2
```

Problem 4

1. **Aim: 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.**

2. **Code:**

```
#include <iostream>
#include <vector>
#include <deque>
using namespace std;
vector<int> maxSlidingWindow(const vector<int>& nums, int k) {
    vector<int> result;
    if (nums.empty() || k <= 0) return result;
```

```
    deque<int> dq;
    for (int i = 0; i < nums.size(); i++) {
        if (!dq.empty() && dq.front() < i - k + 1) {
            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);
    cout << "Maximum values in each sliding window: ";
    for (int maxVal : result) {
        cout << maxVal << " ";
    }
    cout << endl;
    return 0;
}
```

3. Output:

```
Maximum values in each sliding window: 3 3 5 5 6 7
```

Problem 5

1. **Aim: WAP to implement a stack using array and linked list include operations like push pop peak isempty and isfully**

2. **Code:**

```
class ArrayStack {
private:
    int top;
    int maxSize;
    int* stackArray;
public:
    ArrayStack(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1;
    }
    ~ArrayStack() {
        delete[] stackArray;
    }
    void push(int value) {
        if (isFull()) {
            cout << "Stack is full. Cannot push " << value << endl;
            return;
        }
        stackArray[++top] = value;
    }
    int pop() {
        if (isEmpty()) {
            cout << "Stack is empty. Cannot pop." << endl;
            return -1; // or throw an exception
        }
        return stackArray[top--];
    }
    int peek() {
```

```
        if (isEmpty()) {
            cout << "Stack is empty. Cannot peek." << endl;
            return -1; // or throw an exception
        }
        return stackArray[top];
    }
    bool isEmpty() {
        return top == -1;
    }
    bool isFull() {
        return top == maxSize - 1;
    }
};
int main() {
    ArrayStack stack(5);
    stack.push(10);
    stack.push(20);
    stack.push(30);
    cout << "Top element is: " << stack.peek() << endl;
    cout << "Popped element is: " << stack.pop() << endl;
    cout << "Top element is: " << stack.peek() << endl;
    return 0;
}
class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class LinkedListStack {
private:
    Node* top;
```

```
public:
    LinkedListStack() {
        top = nullptr;
    }
    ~LinkedListStack() {
        while (!isEmpty()) {
            pop();
        }
    }
    void push(int value) {
        Node* newNode = new Node(value);
        newNode->next = top;
        top = newNode;
    }
    int pop() {
        if (isEmpty()) {
            cout << "Stack is empty. Cannot pop." << endl;
            return -1; // or throw an exception
        }
        Node* temp = top;
        int poppedValue = top->data;
        top = top->next;
        delete temp;
        return poppedValue;
    }
    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty. Cannot peek." << endl;
            return -1; // or throw an exception
        }
        return top->data;
    }
    bool isEmpty() {
        return top == nullptr;
    }
```

```
};
int main() {
    LinkedListStack stack;
    stack.push(10);
    stack.push(20);
    stack.push(30);
    cout << "Top element is: " << stack.peek() << endl; // 30
    cout << "Popped element is: " << stack.pop() << endl; // 30
    cout << "Top element is: " << stack.peek() << endl; // 20
    return 0;
}
```

3. **Output:**

```
Top element is: 30
Popped element is: 30
Top element is: 20
```

Problem 6

1. **Aim: given a string use the stack to reverse the string**
2. **Code:**

```
string reverseString(const string& str) {
    stack<char> charStack;
    for (char ch : str) {
        charStack.push(ch);
    }
    string reversedStr;
    while (!charStack.empty()) {
        reversedStr += charStack.top();
        charStack.pop();
    }
    return reversedStr;
}
int main() {
```

```
    string input;
    cout << "Enter a string to reverse: ";
    getline(cin, input);
    string reversed = reverseString(input);
    cout << "Reversed string: "<<reversed<<endl;
    return 0;
}
```

3. **Output:**

```
Enter a string to reverse: ABHISHEK
Reversed string: KEHSIHBA
```

Problem 7

1. **Aim: implementation of stack using two queue**
2. **Code:**

```
class StackUsingQueues {
private:
    queue<int> q1;
    queue<int> q2;
public:
    void push(int x) {
        q2.push(x);
        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }
        swap(q1, q2);
    }
    void pop() {
        if (!q1.empty()) {
            q1.pop();
        }
```

```
    }
    int top() {
        if (!q1.empty()) {
            return q1.front();
        }
        throw out_of_range("Stack is empty");
    }
    bool isEmpty() {
        return q1.empty();
    }
    int size() {
        return q1.size();
    }
};
int main() {
    StackUsingQueues stack;
    stack.push(1);
    stack.push(2);
    stack.push(3);
    cout <<"Top element: "<<stack.top()<<endl;
    stack.pop();
    cout <<"Top element after pop: "<<stack.top()<<endl;
    cout<<"Is stack empty? "<<(stack.isEmpty() ? "Yes" : "No")<<endl;
    stack.pop();
    stack.pop();
    cout<<"Is stack empty after popping all elements? "<<(stack.isEmpty()
? "Yes" : "No")<<endl;
    return 0;
}
```

3. Output:

```
Top element: 3
Top element after pop: 2
Is stack empty? No
Is stack empty after popping all elements? Yes
```

Problem 8

1. **Aim: GIVEN A STRING find the first non-repeating character and return its index value, if it does not exist return -1 value**
2. **Code:**

```
int firstNonRepeatingCharacter(const string& str) {
    unordered_map<char, int> charCount;
    for (char ch : str) {
        charCount[ch]++;
    }
    for (int i = 0; i < str.length(); i++) {
        if (charCount[str[i]] == 1) {
            return i;
        }
    }
    return -1;
}
int main() {
    string input;
    cout << "Enter a string: ";
    getline(cin, input);
    int index = firstNonRepeatingCharacter(input);
    if (index != -1) {
        cout << "The first non-repeating character is at index: "<<index<<endl;
    } else {
        cout << "No non-repeating character found." << std::endl;
    }
    return 0;
}
```

3. **Output:**

```
Enter a string: ABHISHEK
The first non-repeating character is at index: 0
```

Problem 9

1. **Aim: CHECK THE MINIMUM VALUE OF STACK AFTER PUSH AND POP OPERATIONS**

2. **Code:**

```
int main()
{
   stack<int> s;
   int arr[] = { 18, 19,29,16,15};
   int n = sizeof(arr)/sizeof(arr[0]);
   for (int i = 0; i < n; i++)
   {
      s.push(arr[i]);

   }
   cout<<" the minimum stack is : "<<arr[0]<<endl;
   return 0;

   }
```

3. **Output:**

```
the minimum stack is : 18
```

Problem 10

1. **Aim: 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.

2. **Code:**

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
using namespace std;
int poisonousPlants(vector<int>& p) {
    int n = p.size();
    vector<int> days(n, 0);
    stack<int> s;

    for (int i = 0; i < n; ++i) {
        while (!s.empty() && p[i] > p[s.top()]) {
            days[i] = max(days[i], days[s.top()] + 1);
            s.pop();
        }
        s.push(i);
    }
    return *max_element(days.begin(), days.end());
}

int main() {
    int n;
    cout << "Enter the number of plants: ";
    cin >> n;
    vector<int> p(n);
    cout << "Enter the pesticide levels: ";
    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;
}
```

3. **Output:**