DOMAIN WINTER WINNING CAMP

DAY: 4

Submitted by: Agrima Sharma

UID: 22BCS15314

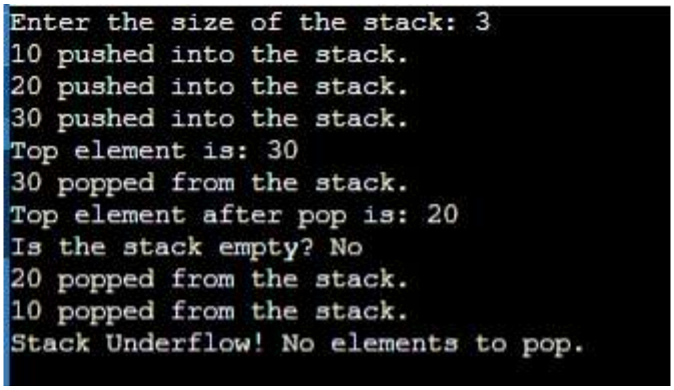
Section: 620/A

1. **Write a C++ program to implement a stack using array including operations like push, pop, peek and isEmpty**

   Code:

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

class Stack {
private:
   int* stackArray;
   int capacity;
   int top;

public:
   // Constructor to initialize the stack
   Stack(int size) {
      capacity = size;
      stackArray = new int[capacity];
      top = -1;
   }

   // Destructor to free allocated memory
   ~Stack() {
      delete[] stackArray;
   }

   // Push operation to add an element to the stack
   void push(int value) {
      if (top == capacity - 1) {
         cout << "Stack Overflow! Cannot push " << value << endl;
      } else {
            stackArray[++top] = value;
            cout << value << " pushed into the stack.\n";
        }
    }

    // Pop operation to remove and return the top element of the stack
    void pop() {
        if (top == -1) {
            cout << "Stack Underflow! No elements to pop.\n";
        } else {
            cout << stackArray[top--] << " popped from the stack.\n";
        }
    }

    // Peek operation to return the top element without removing it
    int peek() {
        if (top == -1) {
            cout << "Stack is empty! No top element.\n";
            return -1; // Return -1 for empty stack
        } else {
            return stackArray[top];
        }
    }

    // Check if the stack is empty
    bool isEmpty() {
        return top == -1;
    }
};

int main() {
    int stackSize;
    cout << "Enter the size of the stack: ";
    cin >> stackSize;

    Stack stack(stackSize);

    // Example operations
    stack.push(10);
    stack.push(20);
    stack.push(30);

    cout << "Top element is: " << stack.peek() << endl;

    stack.pop();
    cout << "Top element after pop is: " << stack.peek() << endl;

    cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

    stack.pop();
    stack.pop();
    stack.pop(); // Trying to pop from an empty stack

    return 0;
}
```

Output



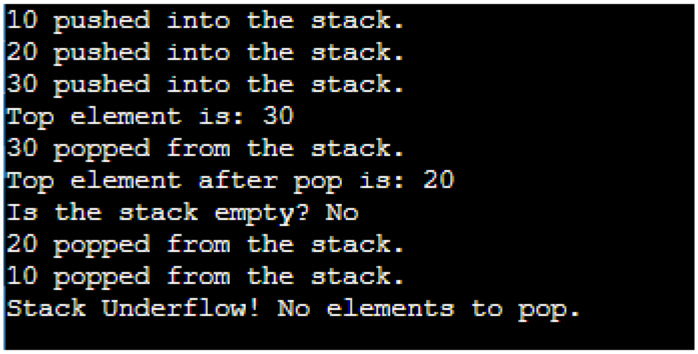
```
Enter the size of the stack: 3
10 pushed into the stack.
20 pushed into the stack.
30 pushed into the stack.
Top element is: 30
30 popped from the stack.
Top element after pop is: 20
Is the stack empty? No
20 popped from the stack.
10 popped from the stack.
Stack Underflow! No elements to pop.
```

2. **Write a C++ program to implement a stack using linked list including operations like push, pop, peek and isEmpty**

Code:

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

// Node structure for the linked list
struct Node {
    int data;
    Node* next;
};

// Stack class using a linked list
class Stack {
private:
    Node* top;

public:
    // Constructor to initialize the stack
    Stack() {
        top = nullptr;
    }

    // Destructor to free memory
    ~Stack() {
        while (top != nullptr) {
            Node* temp = top;
            top = top->next;
            delete temp;
        }
    }

    // Push operation to add an element to the stack
    void push(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        cout << value << " pushed into the stack.\n";
    }

    // Pop operation to remove and return the top element of the stack
    void pop() {
        if (isEmpty()) {
            cout << "Stack Underflow! No elements to pop.\n";
        } else {
            Node* temp = top;
            cout << top->data << " popped from the stack.\n";
            top = top->next;
            delete temp;
        }
    }

    // Peek operation to return the top element without removing it
    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty! No top element.\n";
            return -1; // Return -1 for empty stack
        } else {
            return top->data;
        }
    }

    // Check if the stack is empty
    bool isEmpty() {
        return top == nullptr;
    }
};

int main() {
    Stack stack;

    // Example operations
    stack.push(10);
    stack.push(20);
    stack.push(30);

    cout << "Top element is: " << stack.peek() << endl;

    stack.pop();
    cout << "Top element after pop is: " << stack.peek() << endl;

    cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

    stack.pop();
    stack.pop();
    stack.pop(); // Trying to pop from an empty stack

    return 0;
}
```

Output:



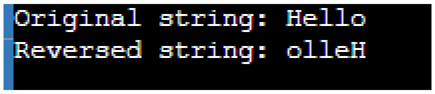
```
10 pushed into the stack.
20 pushed into the stack.
30 pushed into the stack.
Top element is: 30
30 popped from the stack.
Top element after pop is: 20
Is the stack empty? No
20 popped from the stack.
10 popped from the stack.
Stack Underflow! No elements to pop.
```

3. **Given a string, say "Hello". Write a C++ program to reverse this string using stack**

Code:

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

string reverseString(const string& input) {
    stack<char> charStack;

    // Push all characters of the string onto the stack
    for (char ch : input) {
        charStack.push(ch);
    }

    string reversed = "";

    // Pop characters from the stack and build the reversed string
    while (!charStack.empty()) {
        reversed += charStack.top();
        charStack.pop();
    }
    return reversed;
}

int main() {
    string str = "Hello";
    cout << "Original string: " << str << endl;

    string reversedStr = reverseString(str);
    cout << "Reversed string: " << reversedStr << endl;

    return 0;
}
```

Output:



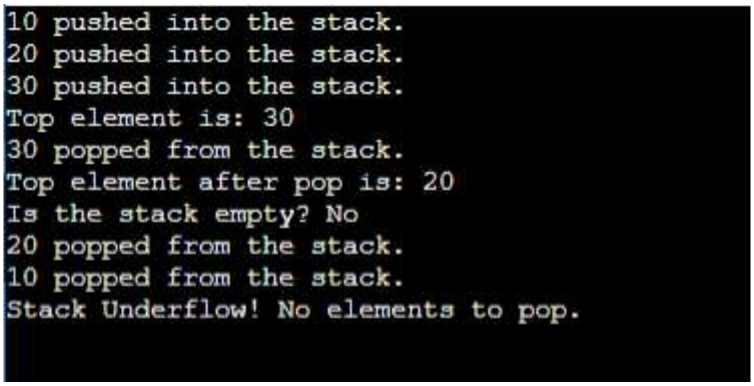
```
Original string: Hello
Reversed string: olleH
```

4. **Write a C++ program for stack implementation using two queues**

Code:

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

class Stack {
private:
    queue<int> q1, q2; // Two queues to simulate stack operations

public:
    // Push operation to add an element to the stack
    void push(int value) {
        q2.push(value); // Push the new element into the second queue

        // Move all elements from q1 to q2
        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }

        // Swap the names of the queues
        swap(q1, q2);
        cout << value << " pushed into the stack.\n";
    }

    // Pop operation to remove the top element of the stack
    void pop() {
        if (q1.empty()) {
            cout << "Stack Underflow! No elements to pop.\n";
        } else {
            cout << q1.front() << " popped from the stack.\n";
            q1.pop();
        }
    }

    // Peek operation to return the top element without removing it
    int peek() {
        if (q1.empty()) {
            cout << "Stack is empty! No top element.\n";
            return -1; // Return -1 for empty stack
        } else {
            return q1.front();
        }
    }

    // Check if the stack is empty
    bool isEmpty() {
        return q1.empty();
    }
};

int main() {
    Stack stack;

    // Example operations
    stack.push(10);
    stack.push(20);
    stack.push(30);
    cout << "Top element is: " << stack.peek() << endl;

    stack.pop();
    cout << "Top element after pop is: " << stack.peek() << endl;

    cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

    stack.pop();
    stack.pop();
    stack.pop(); // Trying to pop from an empty stack

    return 0;
}
```

Output:



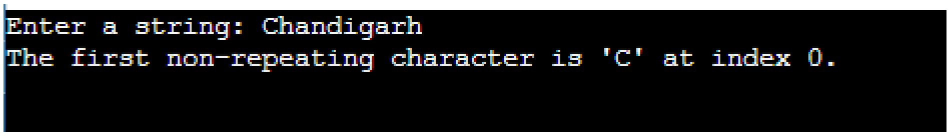
```
10 pushed into the stack.
20 pushed into the stack.
30 pushed into the stack.
Top element is: 30
30 popped from the stack.
Top element after pop is: 20
Is the stack empty? No
20 popped from the stack.
10 popped from the stack.
Stack Underflow! No elements to pop.
```

5. **Given a string. Find the first non-repeating character in it and return its index value. If it does not exist, return -1**

Code:

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

int firstNonRepeatingCharacter(const string& str) {
    const int CHAR_COUNT = 256; // Total number of possible characters
    int charCount[CHAR_COUNT] = {0};

    // Step 1: Count the frequency of each character in the string
    for (char ch : str) {
        charCount[ch]++;
    }
    // Step 2: Find the first character with a frequency of 1
    for (int i = 0; i < str.length(); i++) {
        if (charCount[str[i]] == 1) {
            return i; // Return the index of the first non-repeating character
        }
    }

    return -1; // If no non-repeating character is found
}

int main() {
    string input;
    cout << "Enter a string: ";
    cin >> input;

    int index = firstNonRepeatingCharacter(input);

    if (index != -1) {
        cout << "The first non-repeating character is '" << input[index]
             << "' at index " << index << ".\n";
    } else {
        cout << "No non-repeating character found in the string.\n";
    }

    return 0;
}
```

Output:



```
Enter a string: Chandigarh
The first non-repeating character is 'C' at index 0.
```

6. **Write a C++ program to check the minimum value in the stack**

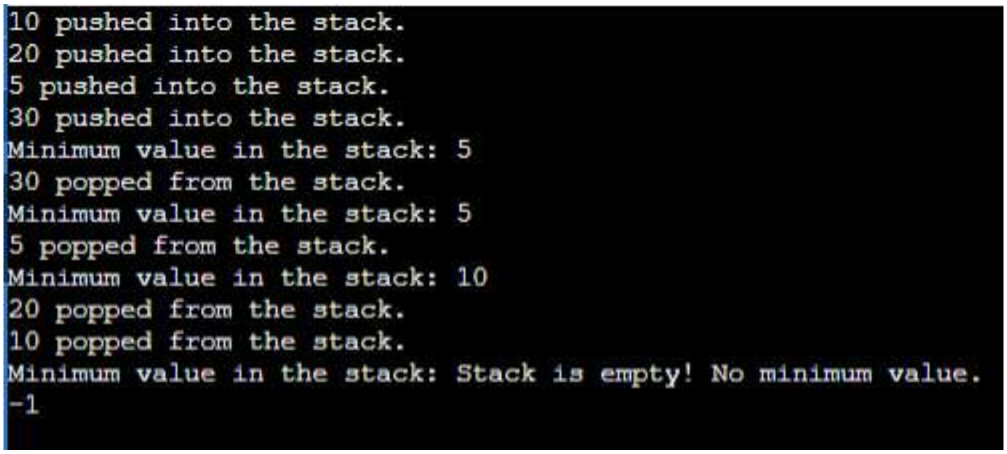
   Code:

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

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

public:
   // Push operation
   void push(int value) {
      mainStack.push(value);
      if (minStack.empty() || value <= minStack.top()) {
         minStack.push(value); // Update minStack if value is smaller or equal to the
current minimum
      }
      cout << value << " pushed into the stack.\n";
   }

   // Pop operation
   void pop() {
      if (mainStack.empty()) {
         cout << "Stack Underflow! No elements to pop.\n";
         return;
      }

      int poppedValue = mainStack.top();
      mainStack.pop();
      if (poppedValue == minStack.top()) {
         minStack.pop(); // Remove from minStack if it was the minimum
      }
      cout << poppedValue << " popped from the stack.\n";
   }

   // Get the minimum value
   int getMin() {
      if (minStack.empty()) {
         cout << "Stack is empty! No minimum value.\n";
         return -1; // Return -1 for empty stack
      }
      return minStack.top();
```

```
    }

    // Check if the stack is empty
    bool isEmpty() {
        return mainStack.empty();
    }
};

int main() {
    MinStack stack;

    stack.push(10);
    stack.push(20);
    stack.push(5);
    stack.push(30);

    cout << "Minimum value in the stack: " << stack.getMin() << endl;

    stack.pop();
    cout << "Minimum value in the stack: " << stack.getMin() << endl;

    stack.pop();
    cout << "Minimum value in the stack: " << stack.getMin() << endl;

    stack.pop();
    stack.pop();
    cout << "Minimum value in the stack: " << stack.getMin() << endl;

    return 0;
}
```

Output:



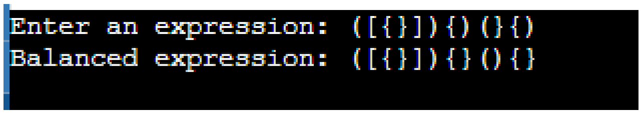
```
10 pushed into the stack.
20 pushed into the stack.
5 pushed into the stack.
30 pushed into the stack.
Minimum value in the stack: 5
30 popped from the stack.
Minimum value in the stack: 5
5 popped from the stack.
Minimum value in the stack: 10
20 popped from the stack.
10 popped from the stack.
Minimum value in the stack: Stack is empty! No minimum value.
-1
```

7. **Write a C++ program to balance the parenthesis by using stack**

Code:

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

// Function to check and balance parentheses
string balanceParentheses(const string& expression) {
    stack<char> s;
    string result = "";

    // Traverse the expression
    for (char ch : expression) {
        // If opening bracket, push onto stack
        if (ch == '(' || ch == '{' || ch == '[') {
            s.push(ch);
            result += ch; // Add to result
        }
        // If closing bracket
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (!s.empty()) {
                char top = s.top();
                // Check if it matches the top of the stack
                if ((ch == ')' && top == '(') ||
                    (ch == '}' && top == '{') ||
                    (ch == ']' && top == '[')) {
                    s.pop(); // Matching pair found
                    result += ch; // Add to result
                } else {
                    // If mismatched, insert the correct closing bracket
                    result += (top == '(' ? ')' : top == '{' ? '}' : ']');
                    s.pop();
                }
            } else {
                // If no opening bracket for the closing one, insert a matching opening
                result = (ch == ')' ? '(' : ch == '}' ? '{' : '[') + result;
                result += ch;
            }
        } else {
            // Add non-bracket characters to the result directly
            result += ch;
        }
    }

    // Append closing brackets for any remaining unmatched opening brackets
    while (!s.empty()) {
        result += (s.top() == '(' ? ')' : s.top() == '{' ? '}' : ']');
        s.pop();
    }

    return result;
}

int main() {
    string expression;
    cout << "Enter an expression: ";
    cin >> expression;

    string balancedExpression = balanceParentheses(expression);

    cout << "Balanced expression: " << balancedExpression << endl;

    return 0;
}
```

Output:



```
Enter an expression: ([{}]){)(}{)
Balanced expression: ([{}]){}(){}
```

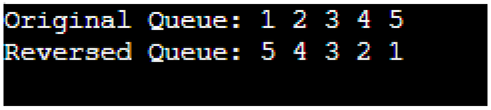
8. **Given a queue. Write a recursive function to reverse it in C++**

Code:

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

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

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

    // Step 2: Recur for the remaining queue
    reverseQueue(q);

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

int main() {
    queue<int> q;

    // Input queue elements
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);
    q.push(5);

    cout << "Original Queue: ";
    queue<int> temp = q; // Temporary queue to display contents
    while (!temp.empty()) {
        cout << temp.front() << " ";
        temp.pop();
    }
    cout << endl;

    // Reverse the queue
    reverseQueue(q);
    cout << "Reversed Queue: ";
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    cout << endl;

    return 0;
}
```

Output:



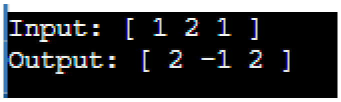
```
Original Queue: 1 2 3 4 5
Reversed Queue: 5 4 3 2 1
```

9. **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.**

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> s;            // Monotonic stack to store indices

    // Traverse the array twice (to handle circular nature)
    for (int i = 0; i < 2 * n; ++i) {
        int currentIndex = i % n; // Circular index

        // Process elements in the stack
        while (!s.empty() && nums[s.top()] < nums[currentIndex]) {
            result[s.top()] = nums[currentIndex];
            s.pop();
        }

        // Push the index onto the stack (only for first traversal)
        if (i < n) {
            s.push(currentIndex);
        }
    }

    return result;
}

int main() {
    vector<int> nums = {1, 2, 1};
    vector<int> result = nextGreaterElements(nums);

    cout << "Input: [ ";
    for (int num : nums) {
        cout << num << " ";
    }
    cout << "]\n";

    cout << "Output: [ ";
    for (int val : result) {
        cout << val << " ";
    }
    cout << "]\n";

    return 0;
}
```

Output:



```
Input: [ 1 2 1 ]
Output: [ 2 -1 2 ]
```

10. **Suppose there is a circle. There are N petrol pumps on that circle. Petrol pumps are numbered 0 to (N-1) (both inclusive). You have two pieces of information corresponding to each of the petrol pump: (1) the amount of petrol that particular petrol pump will give, and (2) the distance from that petrol pump to the next petrol pump.**

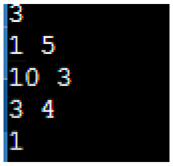
Code:

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

```
using namespace std;

int findStartingPoint(int N, vector<pair<int, int>>& pumps) {
   int start = 0;    // Start index
   long long currentFuel = 0; // Current fuel in the tank
   long long totalFuel = 0;   // Total fuel for the entire journey

   // Traverse each petrol pump
   for (int i = 0; i < N; ++i) {
      int petrol = pumps[i].first;
      int distance = pumps[i].second;

      // Add petrol at the current pump and subtract the distance to the next pump
      currentFuel += petrol - distance;
      totalFuel += petrol - distance;

      // If currentFuel becomes negative, reset the starting point to i + 1
      if (currentFuel < 0) {
         start = i + 1;
         currentFuel = 0;  // Reset current fuel
      }
   }

   // If total fuel is non-negative, return the starting point
   return (totalFuel >= 0) ? start : -1;  // If total fuel is negative, return -1 (impossible
to complete)
}

int main() {
   int N;
   cin >> N;

   vector<pair<int, int>> pumps(N);

   // Input petrol pump details (petrol provided, distance to next pump)
   for (int i = 0; i < N; ++i) {
      cin >> pumps[i].first >> pumps[i].second;
   }
```

```
    // Find and output the first valid starting point
    int result = findStartingPoint(N, pumps);
    cout << result << endl;

    return 0;
}
```

Output:



```
3
1 5
10 3
3 4
1
```