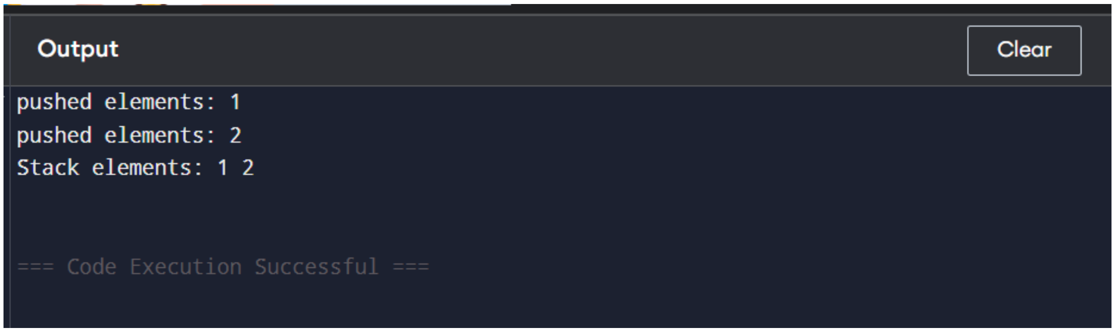
## DOMAIN WINTER WINNING CAMP

**Student Name:** Mannat Gupta
**Branch:** BE-CSE
**Semester:** 5th

**UID:** 22BCS15281
**Section/Group:** 620-A
**Date of Performance:** 24/12/24

### 1. Push operation in stack

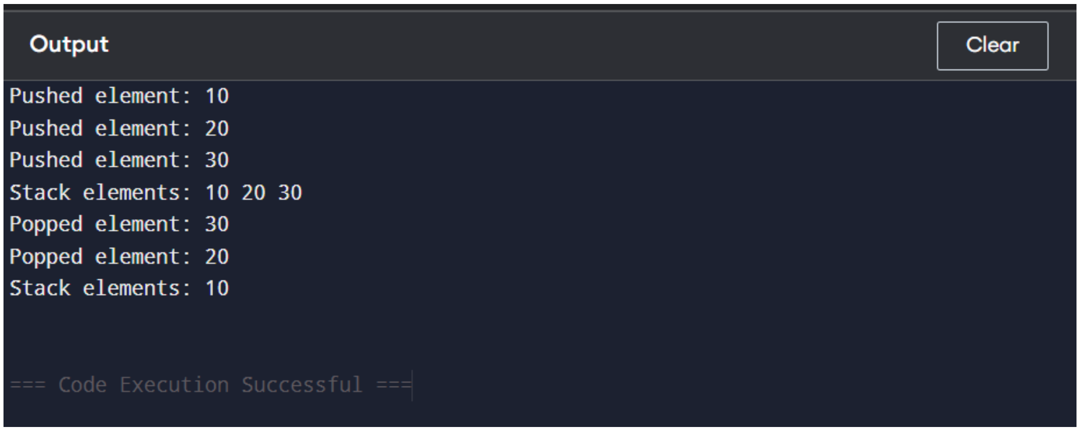
```
#include <iostream>
using namespace std;

const int maxsize=100;

class Stack{
    private:
    int arr[maxsize];
    int top;

    public:
    Stack(){
        top=-1;
    }

void push(int x){
    if(top==maxsize-1){
        cout<<"Stack overflow"<<endl;
        return;
    }
    top = top + 1;
    arr[top] = x;
    cout<<"pushed elements: "<<x<<endl;
}

void display(){
    if(top==-1){
        cout<<"Stack underflow"<<endl;
        return;
    }
    cout<<"Stack elements: ";
    for(int i=0;i<=top;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}
};

int main() {
    Stack s;
    s.push(1);
    s.push(2);

    s.display();

    return 0;
}
```



```
pushed elements: 1
pushed elements: 2
Stack elements: 1 2


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

## 2. Pop operation in stack

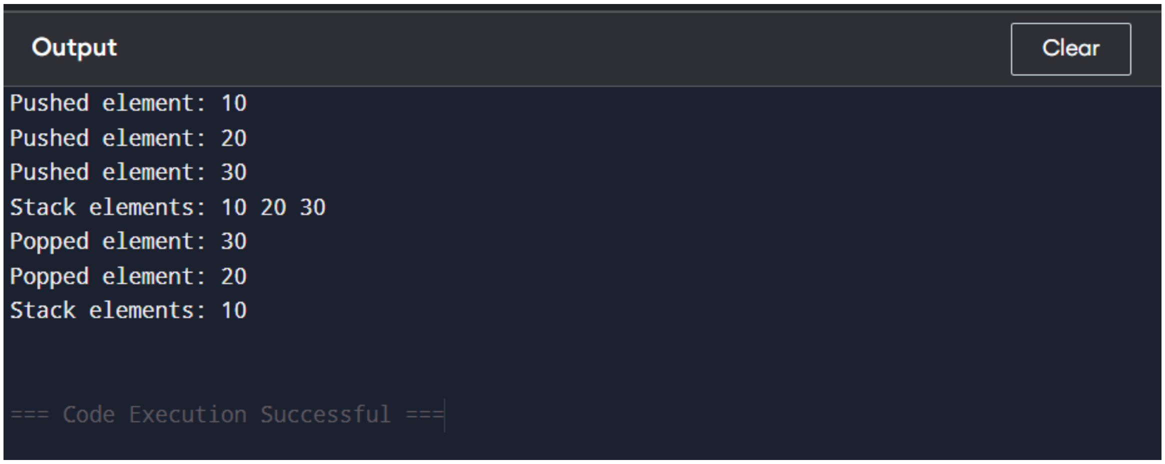
```
#include <iostream>
using namespace std;

const int maxSize = 100;

class Stack {
private:
    int arr[maxSize];
    int top;

public:
    Stack() {
        top = -1;
    }

    void push(int x) {
        if (top == maxSize - 1) {
            cout << "Stack Overflow!" << endl;
            return;
        }
        top = top + 1;
        arr[top] = x;
        cout << "Pushed element: " << x << endl;
    }

    void pop() {
        if (top == -1) {
            cout << "Stack Underflow!" << endl;
            return;
        }
        cout << "Popped element: " << arr[top] << endl;
        top = top - 1;
    }

    void display() {
        if (top == -1) {
            cout << "Stack is empty!" << endl;
            return;
        }
        cout << "Stack elements: ";
        for (int i = 0; i <= top; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.display();
    s.pop();
    s.pop();
    s.display();
    return 0;
}
```



```
Pushed element: 10
Pushed element: 20
Pushed element: 30
Stack elements: 10 20 30
Popped element: 30
Popped element: 20
Stack elements: 10


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

## 3. Stack implementation using linked list

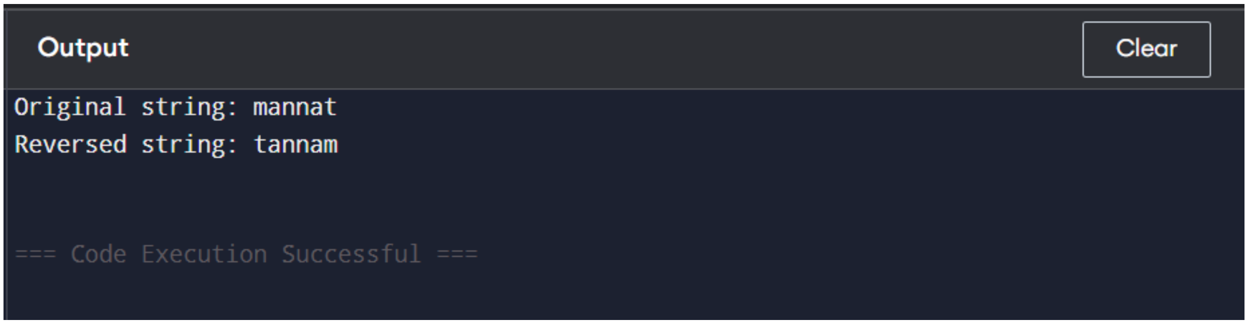
```
#include <iostream>
using namespace std;

struct Node{
    int data;
    Node* next;
};

class Stack{
    private:
    Node* top;
    public:
    Stack(){
        top=nullptr;
    }
void push(int x){
    Node* newNode= new Node();
    newNode->data=x;
    newNode->next=top;
    top=newNode;
}

void pop(){
    if(top==nullptr){
        cout<<"Stack underflow"<<endl;
        return;
    }
    Node* temp=top;
    top=top->next;
    delete temp;
}

void peek() {
    if (top != nullptr) {
        cout << "Top element: " << top->data << endl;
    }
    else {
        cout << "Stack is empty" << endl;
    }
}
bool isEmpty() {
    return top == nullptr;
}
void display(){
    Node* temp=top;
    while(temp!=nullptr){
        cout<<temp->data<<" ";
        temp=temp->next;
    }
    cout<<endl;
    }
};

int main() {
    Stack s;
    cout<<"Stack after pushing: ";
    s.push(1);
    s.push(2);
    s.push(3);
    s.display();
    cout<<endl;

    cout<<"Stack after poping: ";
    s.pop();
    s.pop();
    s.display();
    cout<<endl;

    cout << "Peek operation: ";
    s.peek();
    cout<<endl;

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

    return 0;
}
```



```
Pushed element: 10
Pushed element: 20
Pushed element: 30
Stack elements: 10 20 30
Popped element: 30
Popped element: 20
Stack elements: 10


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

### 4. Reverse a string using stack

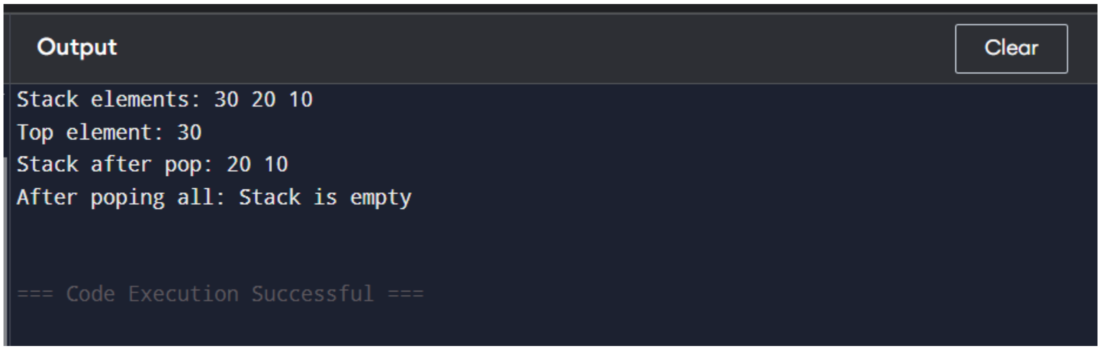
```
#include <iostream>
using namespace std;

class Stack {
private:
    static const int maxsize = 100;
    char stack[maxsize];
    int top;

public:
    Stack() {
        top = -1;
    }

    void push(char ch) {
        if (top == maxsize - 1) return;
        top = top + 1;
        stack[top] = ch;
    }

    char pop() {
        if (top == -1) return '\0';
        char ch = stack[top];
        top = top - 1;
        return ch;
    }

    bool isEmpty() {
        return top == -1;
    }
};

void reverseString(char str[]) {
    Stack s;
    int i = 0;

    while (str[i] != '\0') {
        s.push(str[i]);
        i++;
    }

    i = 0;
    while (!s.isEmpty()) {
        str[i++] = s.pop();
    }
}

int main() {
    char str[] = "mannat";
    cout << "Original string: " << str << endl;
    reverseString(str);
    cout << "Reversed string: " << str << endl;
    return 0;
}
```



```
Original string: mannat
Reversed string: tannam


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

## 5. Print Multiplication Table of a Number

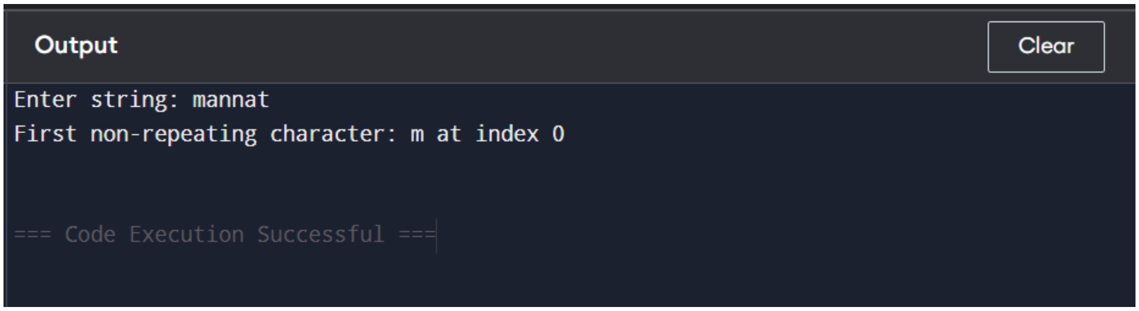
```
#include <iostream>
#include <queue>
using namespace std;

class Stack {
private:
    queue<int> q1, 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()) {
            cout << "Stack is empty\n";
            return;
        }
        q1.pop();
    }

    int top() {
        if (q1.empty()) {
            cout << "Stack is empty\n";
            return -1;
        }
        return q1.front();
    }

    bool isEmpty() {
        return q1.empty();
    }

    void display() {
        if (q1.empty()) {
            cout << "Stack is empty" << endl;
            return;
        }

        queue<int> temp = q1;
        while (!temp.empty()) {
            cout << temp.front() << " ";
            temp.pop();
        }
        cout << endl;
    }
};

int main() {
    Stack stack;

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

    cout << "Stack elements: ";
    stack.display();

    cout << "Top element: " << stack.top() << endl;
    stack.pop();

    cout << "Stack after pop: ";
    stack.display();
    cout << "After poping all: ";
    stack.pop();
    stack.pop();
    stack.pop();

    return 0;
}
```



```
Stack elements: 30 20 10
Top element: 30
Stack after pop: 20 10
After poping all: Stack is empty


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

**6. Given a string, find the first non-repeating character and its index. If no such character exists, return -1.**

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

pair<char, int> firstNonRepeatingChar(const string& str) {
    int freq[26] = {0};
    for (char c : str) {
        freq[c - 'a']++;
    }
    for (int i = 0; i < str.length(); i++) {
        if (freq[str[i] - 'a'] == 1) {
            return {str[i], i};
        }
    }
    return {'-', -1};
}

int main() {
    cout<<"Enter string: ";
    string input;
    cin >> input;
    pair<char, int> result = firstNonRepeatingChar(input);
    if (result.second != -1) {
        cout << "First non-repeating character: " << result.first << " at index " << result.second <<
endl;
    } else {
        cout << "No non-repeating character found." << endl;
    }
    return 0;
}
```

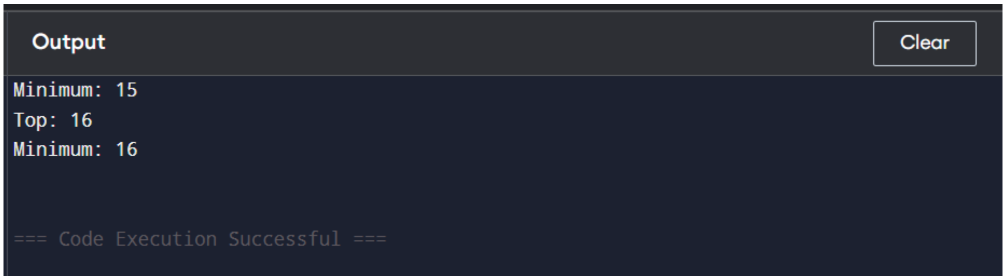


```
Enter string: mannat
First non-repeating character: m at index 0


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

### 7. Minimum stack value

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

using namespace std;

class MinStack {
private:
    stack<int> s;
    stack<int> minStack;

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

    void pop() {
        if (s.top() == minStack.top()) {
            minStack.pop();
        }
        s.pop();
    }

    int top() {
        return s.top();
    }

    int getMin() {
        return minStack.top();
    }
};
```

```
int main() {
    MinStack minStack;

    int values[] = {19, 18, 29, 16, 15};

    for (int val : values) {
        minStack.push(val);
    }

    cout << "Minimum: " << minStack.getMin() << endl;

    minStack.pop();
    cout << "Top: " << minStack.top() << endl;
    cout << "Minimum: " << minStack.getMin() << endl;

    return 0;
}
```

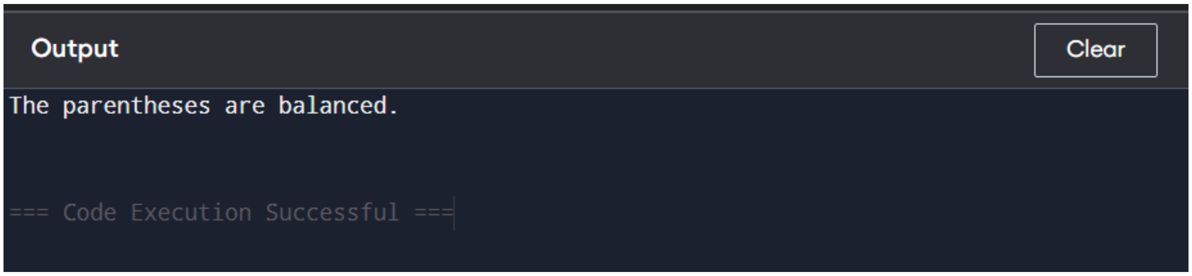


```
Minimum: 15
Top: 16
Minimum: 16


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

## 8. Balance the number of parenthesis using stack

```
#include <iostream>
#include <stack>
#include <string>

using namespace std;

bool isBalanced(const string& expression) {
    stack<char> s;
    for (char ch : expression) {
        if (ch == '(') {
            s.push(ch);
        } else if (ch == ')') {
            if (s.empty()) {
                return false;
            }
            s.pop();
        }
    }
    return s.empty();
}
```

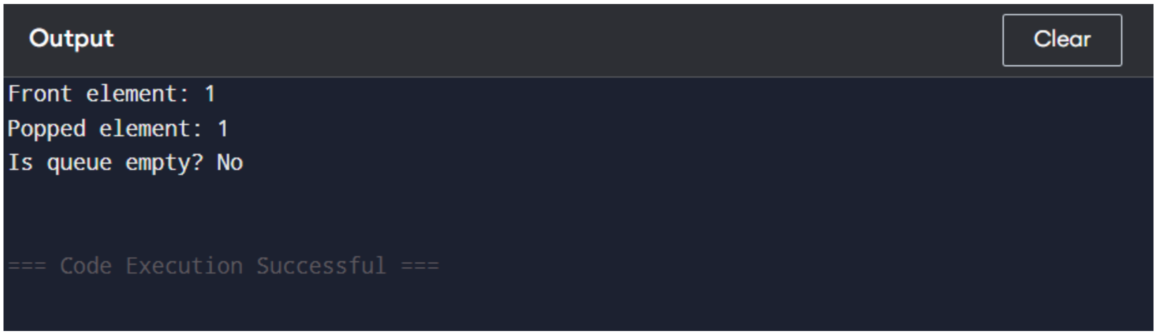
```
int main() {
    string expression = "((19+18)*(29-16))/(15)";
    if (isBalanced(expression)) {
        cout << "The parentheses are balanced." << endl;
    } else {
        cout << "The parentheses are not balanced." << endl;
    }
    return 0;
}
```



```
The parentheses are balanced.


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

**9. Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).**
**Implement the MyQueue class:**
**void push(int x) Pushes element x to the back of the queue.**
**int pop() Removes the element from the front of the queue and returns it.**
**int peek() Returns the element at the front of the queue.**
**boolean empty() Returns true if the queue is empty, false otherwise.**

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

using namespace std;

class MyQueue {
private:
    stack<int> stack1;
    stack<int> stack2;

public:
    void push(int x) {
        stack1.push(x);
    }

    int pop() {
        if (stack2.empty()) {
            while (!stack1.empty()) {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        int front = stack2.top();
        stack2.pop();
        return front;
    }

    int peek() {
        if (stack2.empty()) {
            while (!stack1.empty()) {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        return stack2.top();
    }

    bool empty() {
        return stack1.empty() && stack2.empty();
    }
};

int main() {
    MyQueue queue;

    queue.push(1);
    queue.push(2);
    queue.push(3);

    cout << "Front element: " << queue.peek() << endl;
    cout << "Popped element: " << queue.pop() << endl;
    cout << "Is queue empty? " << (queue.empty() ? "Yes" : "No") << endl;

    return 0;
}
```

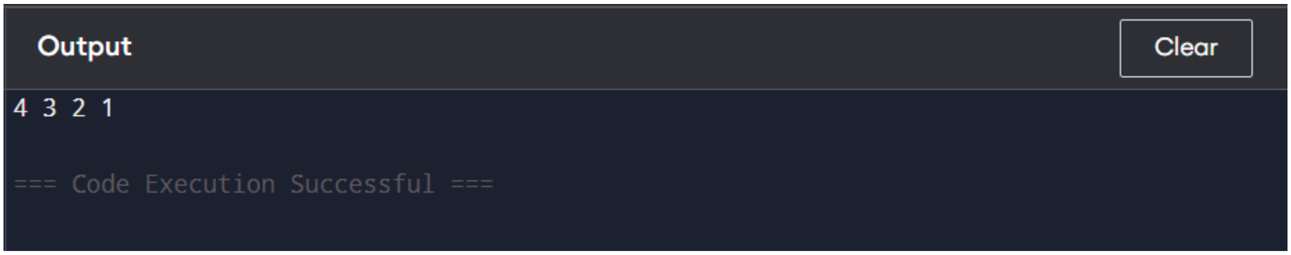


```
Front element: 1
Popped element: 1
Is queue empty? No


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

10. Given a queue, write a recursive function to reverse it. .

```
#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() {
    queue<int> q;

    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);

    reverseQueue(q);

    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }

    return 0;
}
```



```
4 3 2 1

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