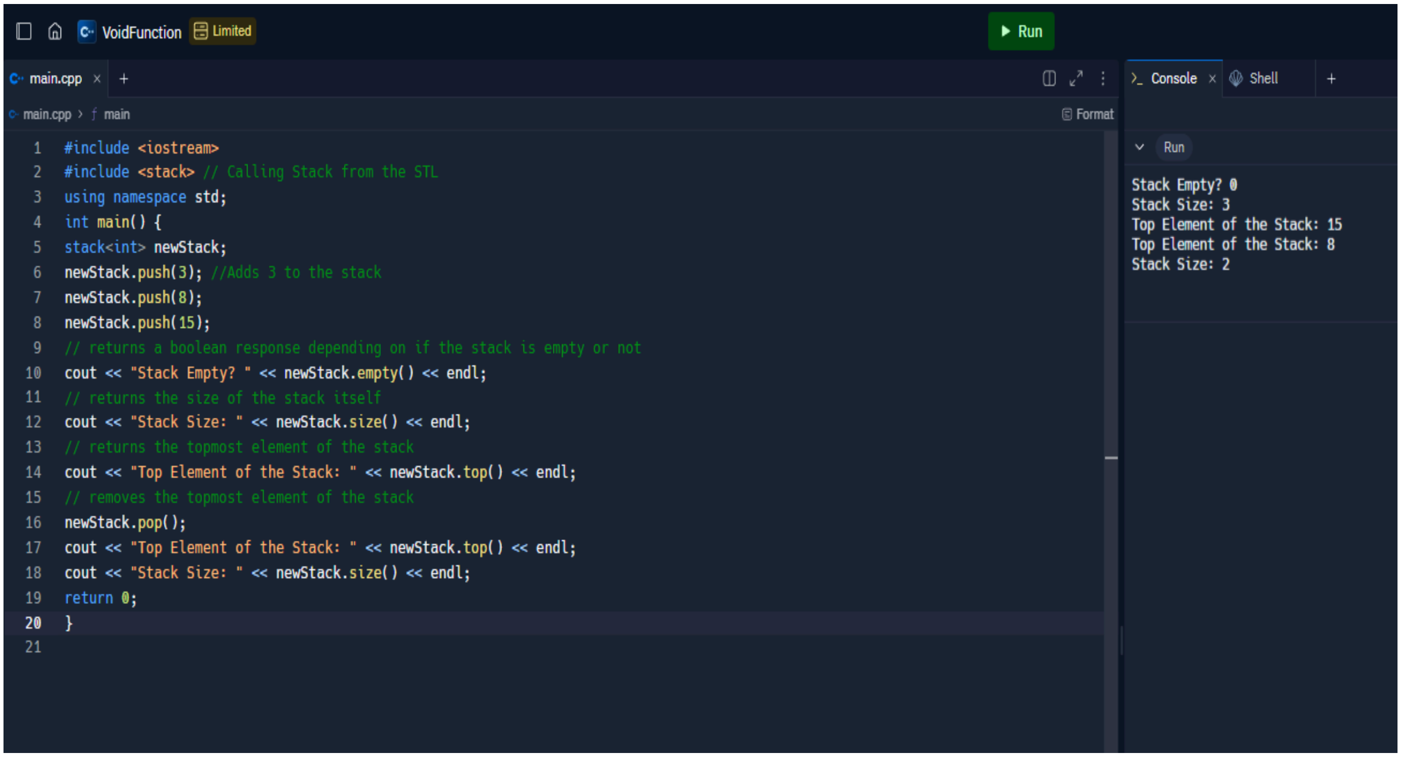
Activity No. 4	
STACKS	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: OCTOBER 4, 2024**
Section: CPE21S1	**Date Submitted: OCTOBER 7, 2024**
Name(s): GASPAR, AARON ROWEN O.	**Instructor: MS. MARIA RIZETTE SAYO**

6. Output

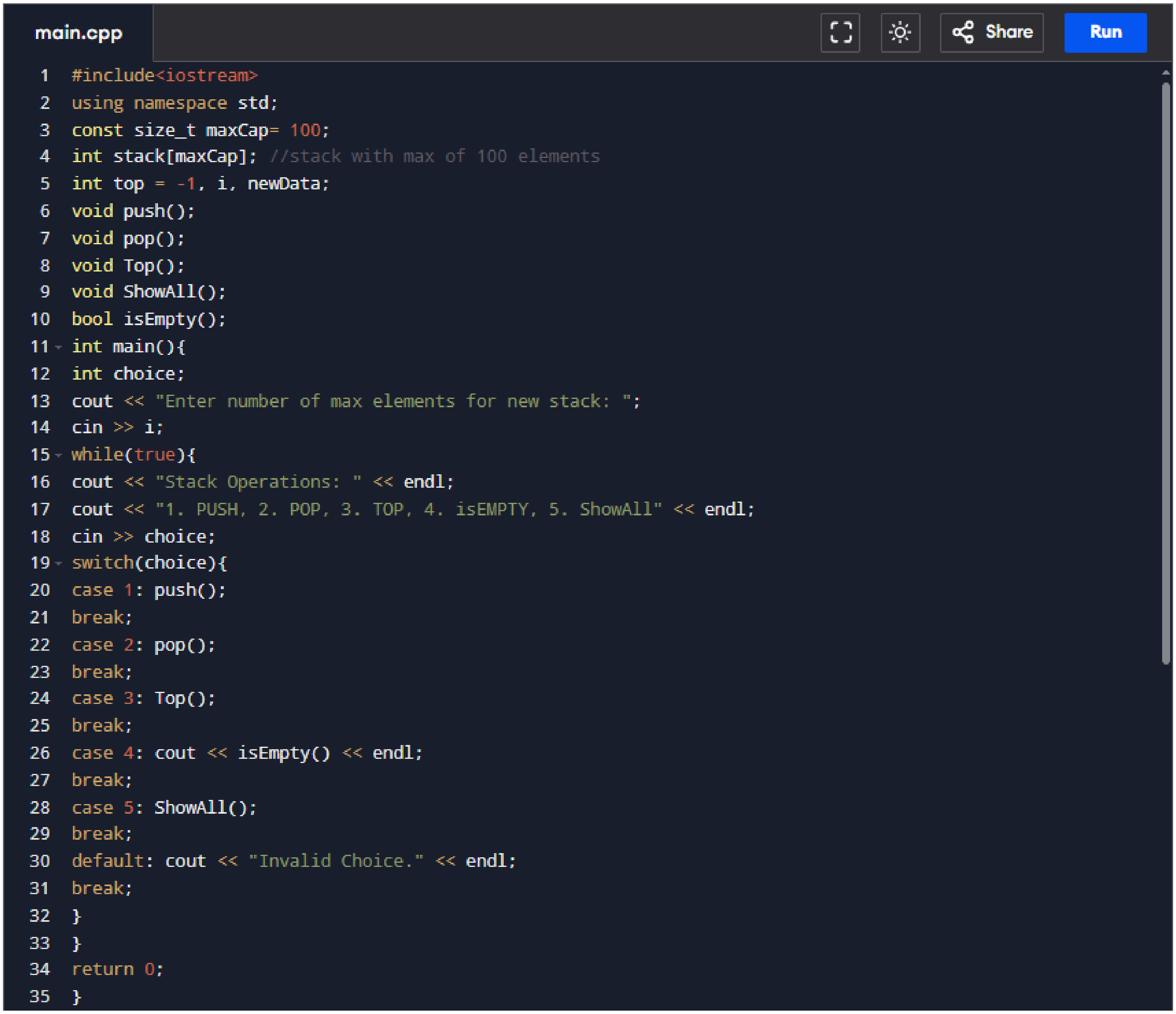
ILO A:



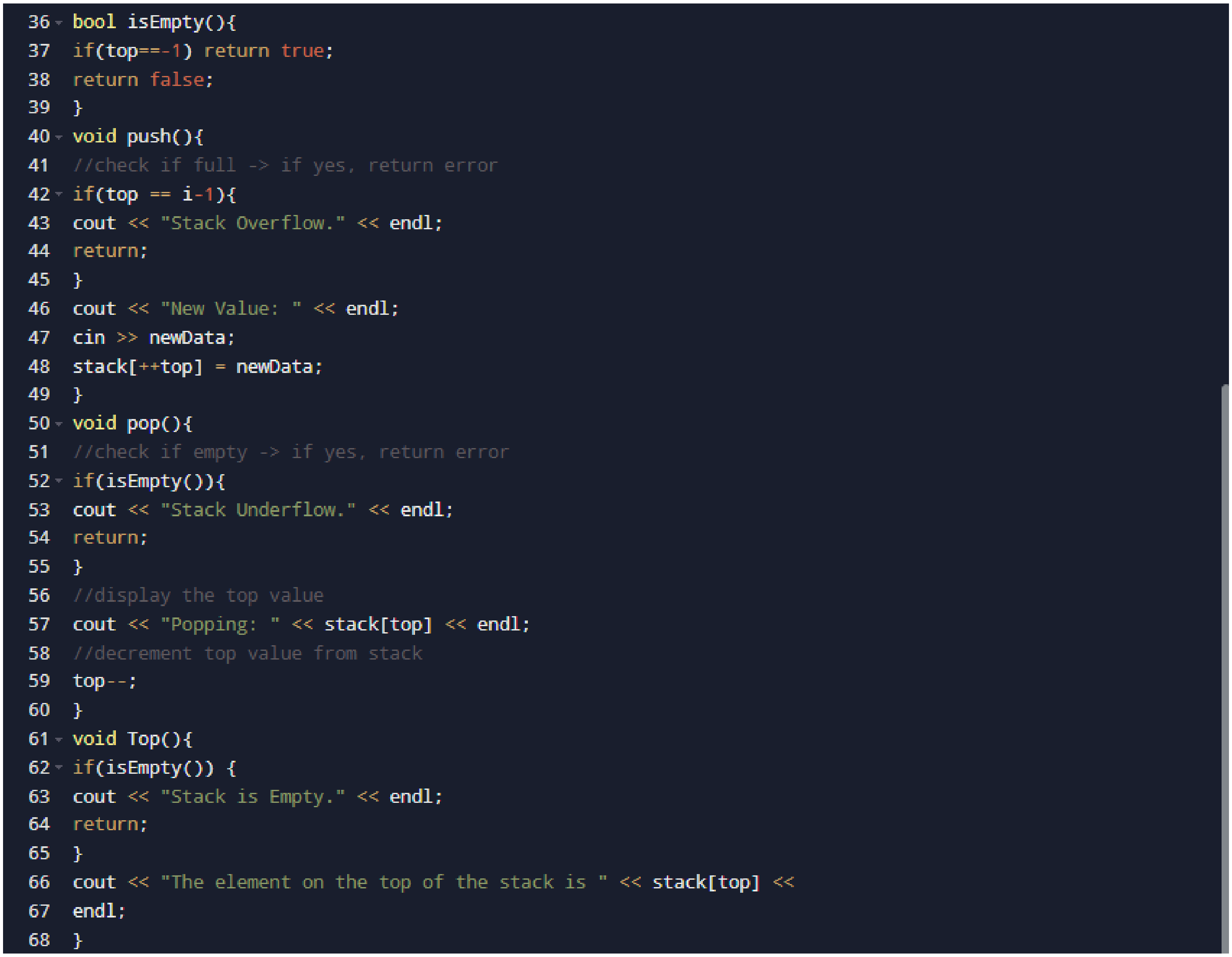
```
#include <iostream>
#include <stack> // Calling Stack from the STL
using namespace std;
int main() {
stack<int> newStack;
newStack.push(3); //Adds 3 to the stack
newStack.push(8);
newStack.push(15);
// returns a boolean response depending on if the stack is empty or not
cout << "Stack Empty? " << newStack.empty() << endl;
// returns the size of the stack itself
cout << "Stack Size: " << newStack.size() << endl;
// returns the topmost element of the stack
cout << "Top Element of the Stack: " << newStack.top() << endl;
// removes the topmost element of the stack
newStack.pop();
cout << "Top Element of the Stack: " << newStack.top() << endl;
cout << "Stack Size: " << newStack.size() << endl;
return 0;
}
```

```
Stack Empty? 0
Stack Size: 3
Top Element of the Stack: 15
Top Element of the Stack: 8
Stack Size: 2
```

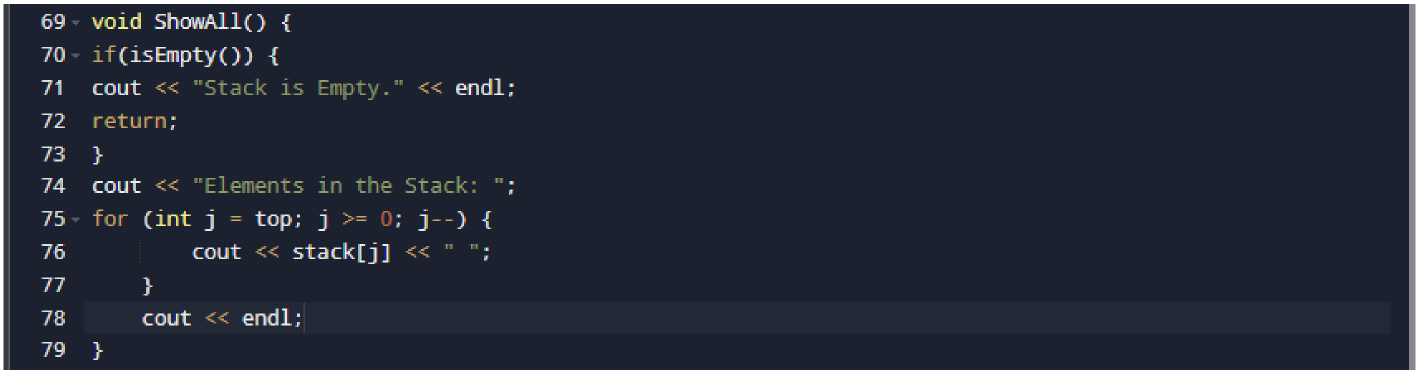
ILO B.1:



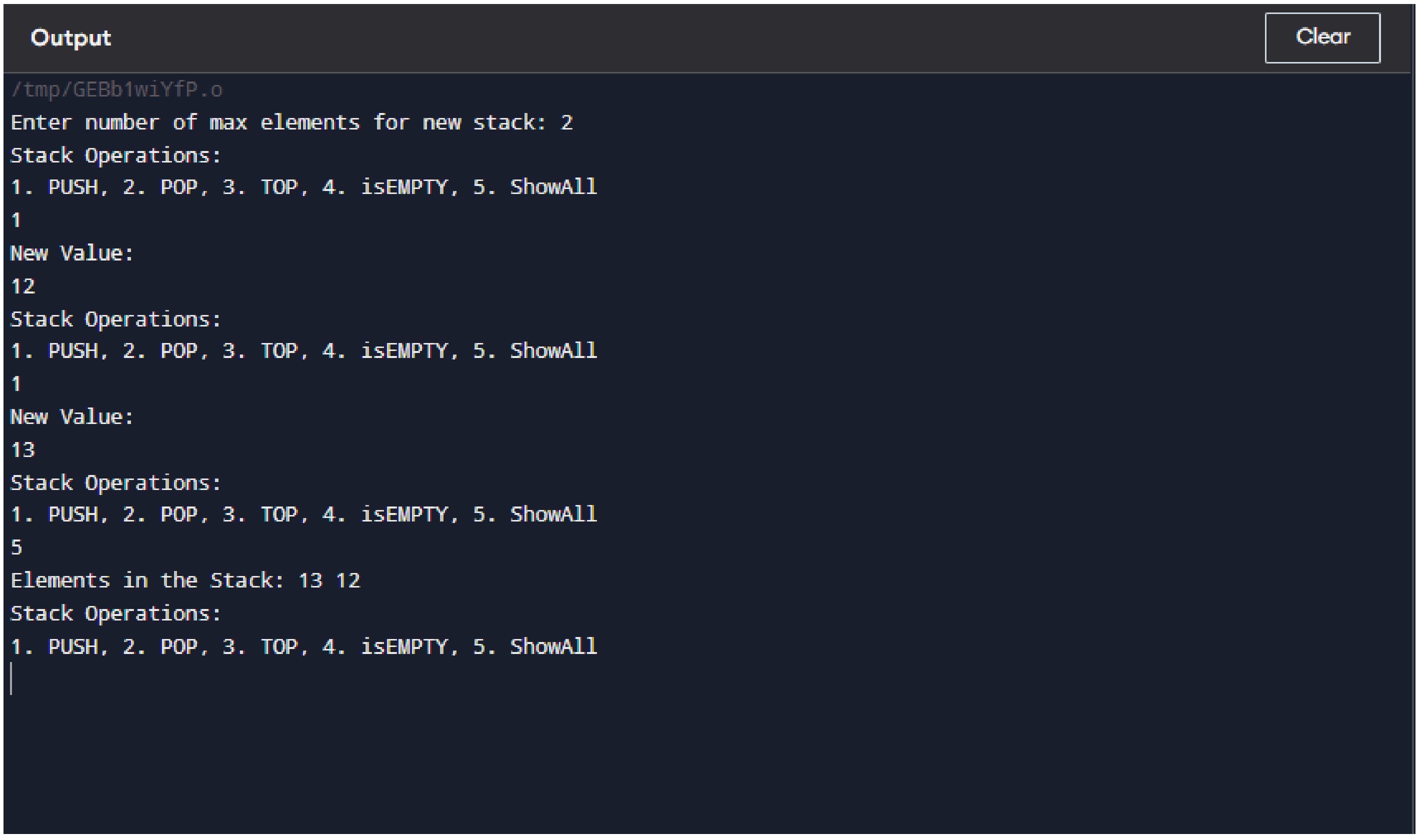
```cpp
#include<iostream>
using namespace std;
const size_t maxCap= 100;
int stack[maxCap]; //stack with max of 100 elements
int top = -1, i, newData;
void push();
void pop();
void Top();
void ShowAll();
bool isEmpty();
int main(){
int choice;
cout << "Enter number of max elements for new stack: ";
cin >> i;
while(true){
cout << "Stack Operations: " << endl;
cout << "1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. ShowAll" << endl;
cin >> choice;
switch(choice){
case 1: push();
break;
case 2: pop();
break;
case 3: Top();
break;
case 4: cout << isEmpty() << endl;
break;
case 5: ShowAll();
break;
default: cout << "Invalid Choice." << endl;
break;
}
}
return 0;
}
```



```
bool isEmpty(){
if(top==-1) return true;
return false;
}
void push(){
//check if full -> if yes, return error
if(top == i-1){
cout << "Stack Overflow." << endl;
return;
}
cout << "New Value: " << endl;
cin >> newData;
stack[++top] = newData;
}
void pop(){
//check if empty -> if yes, return error
if(isEmpty()){
cout << "Stack Underflow." << endl;
return;
}
//display the top value
cout << "Popping: " << stack[top] << endl;
//decrement top value from stack
top--;
}
void Top(){
if(isEmpty()) {
cout << "Stack is Empty." << endl;
return;
}
cout << "The element on the top of the stack is " << stack[top] <<
endl;
}
```

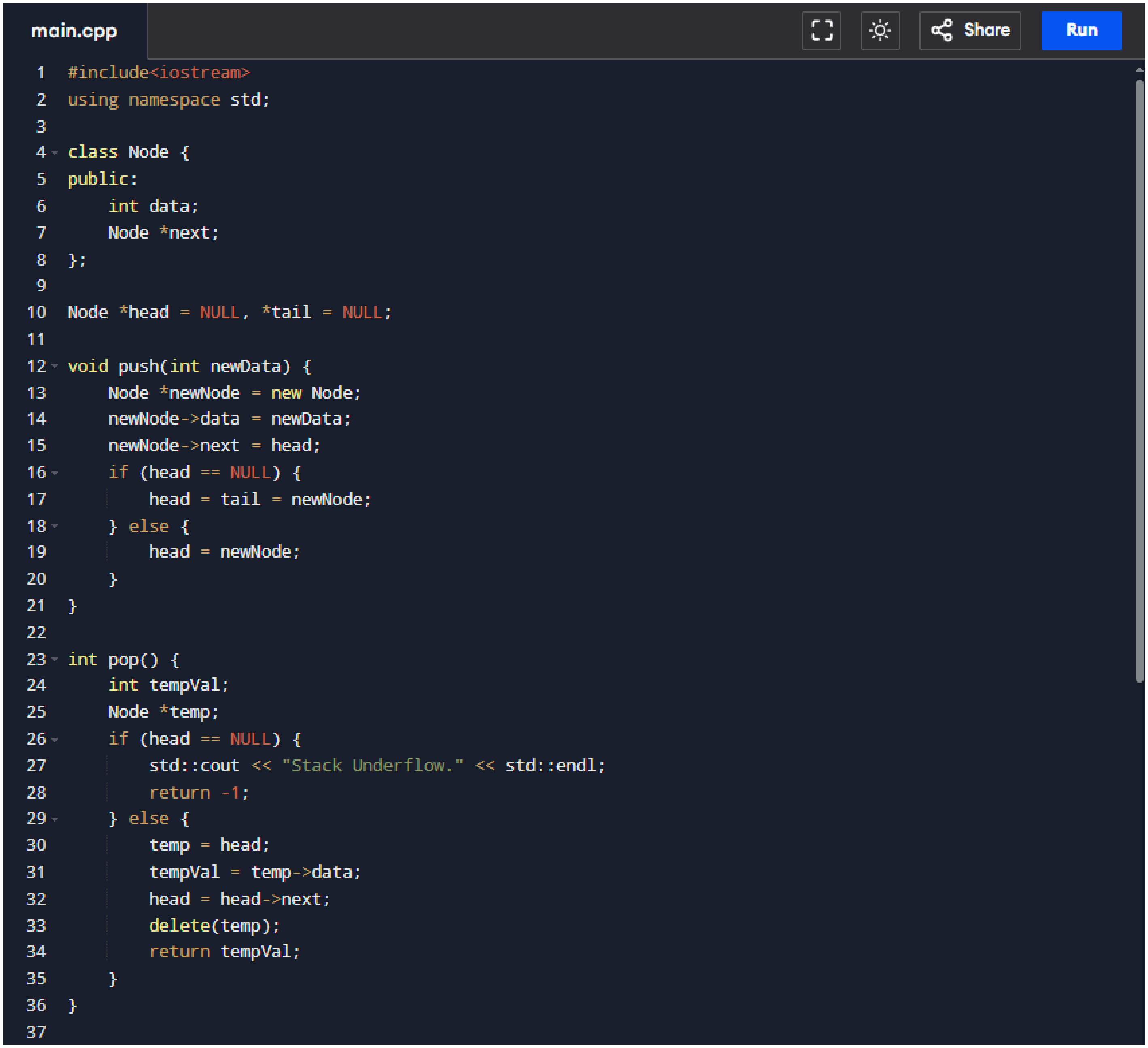


```
void ShowAll() {
if(isEmpty()) {
cout << "Stack is Empty." << endl;
return;
}
cout << "Elements in the Stack: ";
for (int j = top; j >= 0; j--) {
        cout << stack[j] << " ";
    }
    cout << endl;
}
```

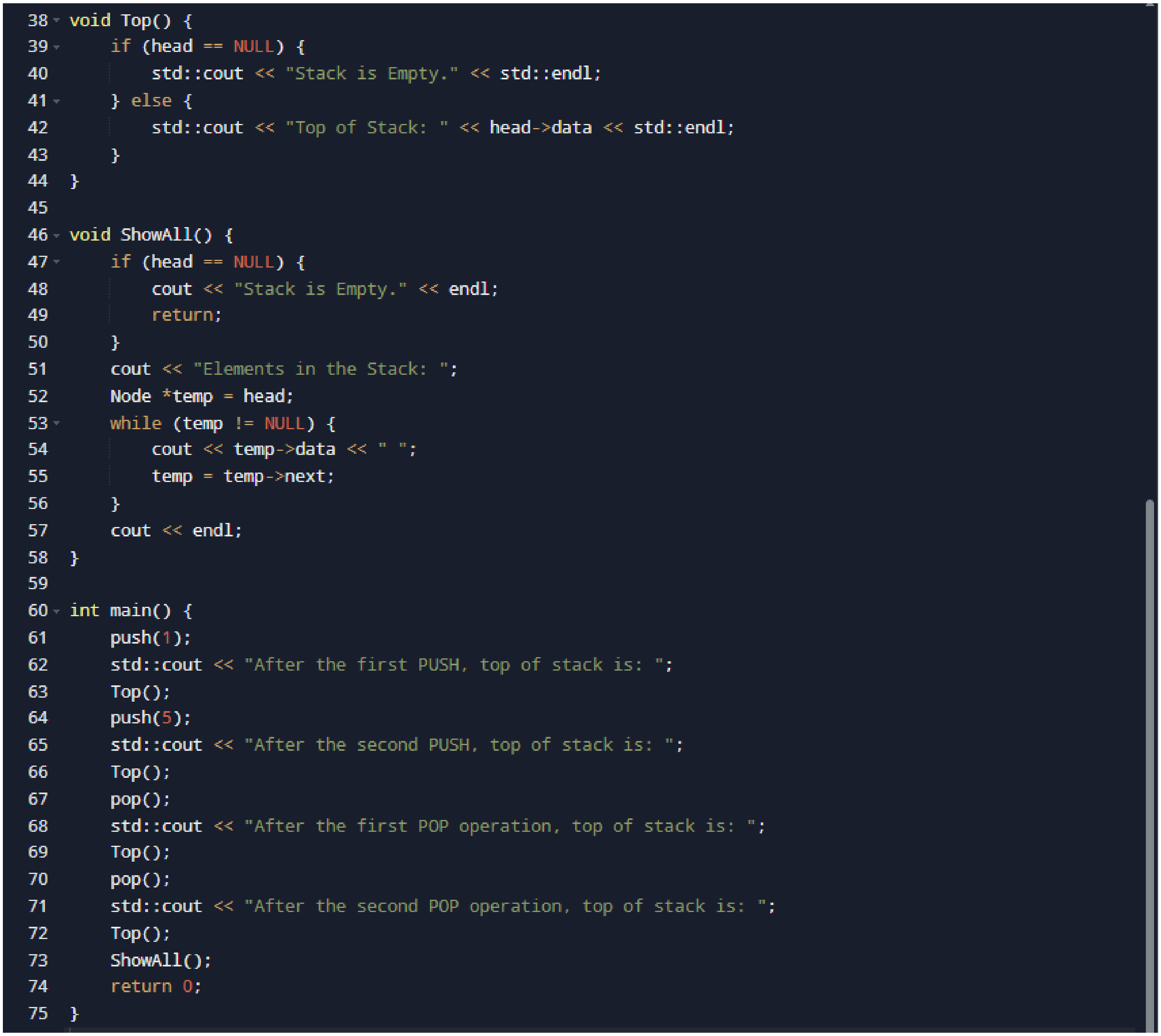


```
/tmp/GEBb1wiYfP.o
Enter number of max elements for new stack: 2
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. ShowAll
1
New Value:
12
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. ShowAll
1
New Value:
13
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. ShowAll
5
Elements in the Stack: 13 12
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. ShowAll
```

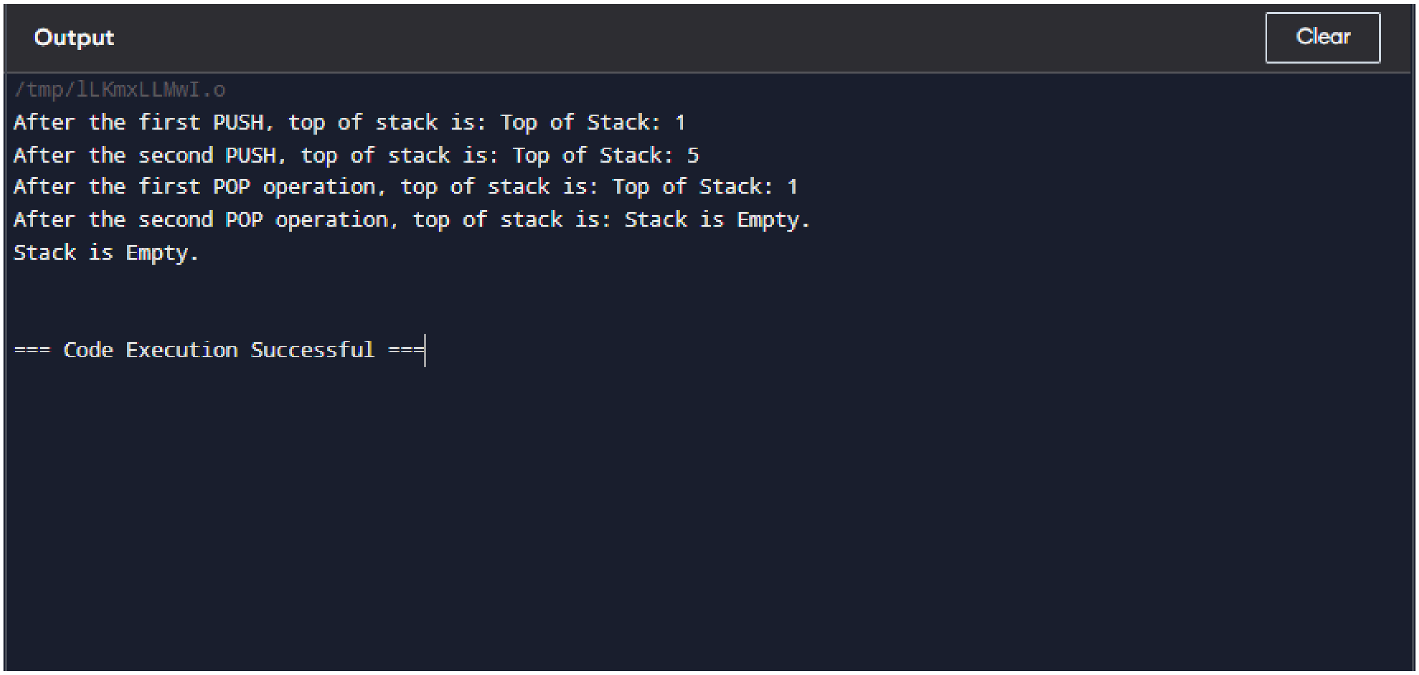
ILO B.2:



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

class Node {
public:
    int data;
    Node *next;
};

Node *head = NULL, *tail = NULL;

void push(int newData) {
    Node *newNode = new Node;
    newNode->data = newData;
    newNode->next = head;
    if (head == NULL) {
        head = tail = newNode;
    } else {
        head = newNode;
    }
}

int pop() {
    int tempVal;
    Node *temp;
    if (head == NULL) {
        std::cout << "Stack Underflow." << std::endl;
        return -1;
    } else {
        temp = head;
        tempVal = temp->data;
        head = head->next;
        delete(temp);
        return tempVal;
    }
}

```



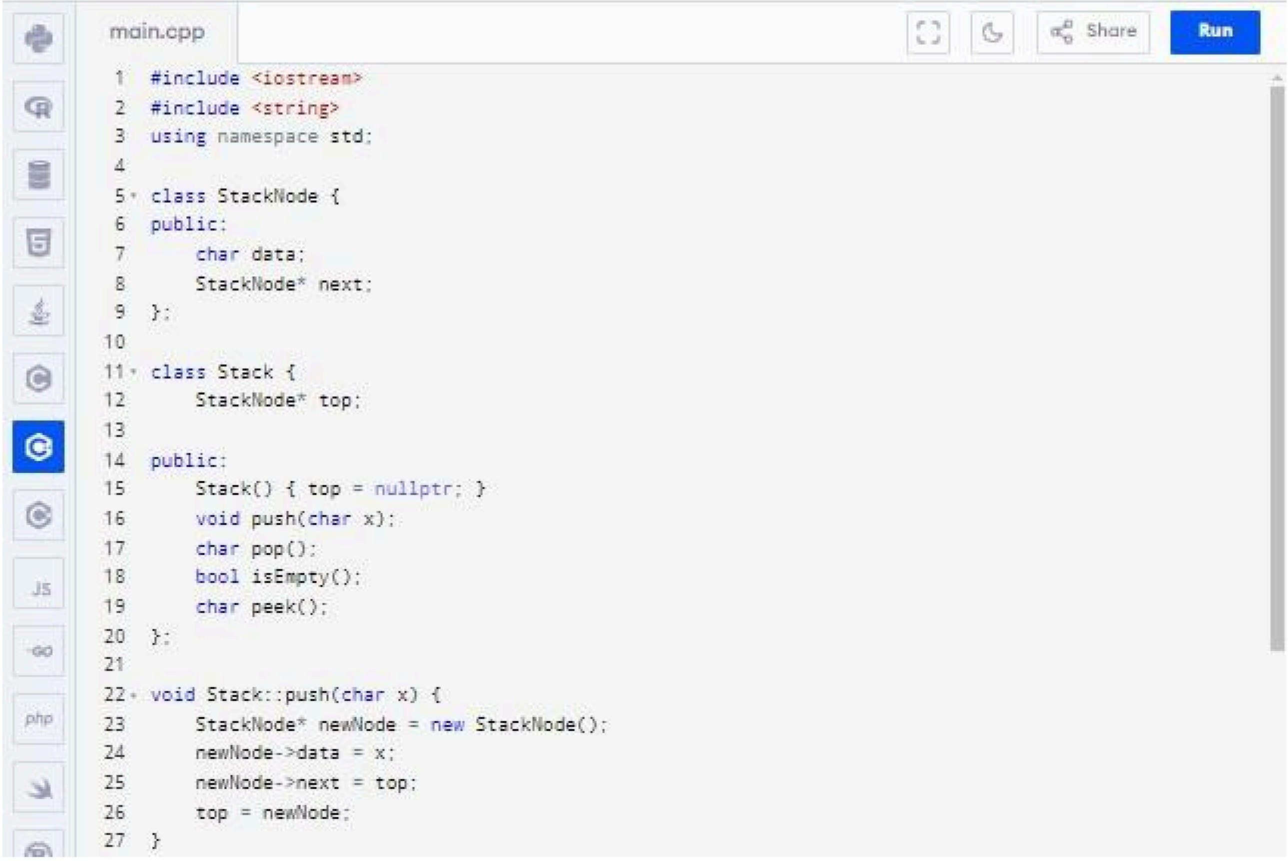
```
void Top() {
    if (head == NULL) {
        std::cout << "Stack is Empty." << std::endl;
    } else {
        std::cout << "Top of Stack: " << head->data << std::endl;
    }
}

void ShowAll() {
    if (head == NULL) {
        cout << "Stack is Empty." << endl;
        return;
    }
    cout << "Elements in the Stack: ";
    Node *temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    push(1);
    std::cout << "After the first PUSH, top of stack is: ";
    Top();
    push(5);
    std::cout << "After the second PUSH, top of stack is: ";
    Top();
    pop();
    std::cout << "After the first POP operation, top of stack is: ";
    Top();
    pop();
    std::cout << "After the second POP operation, top of stack is: ";
    Top();
    ShowAll();
    return 0;
}
```



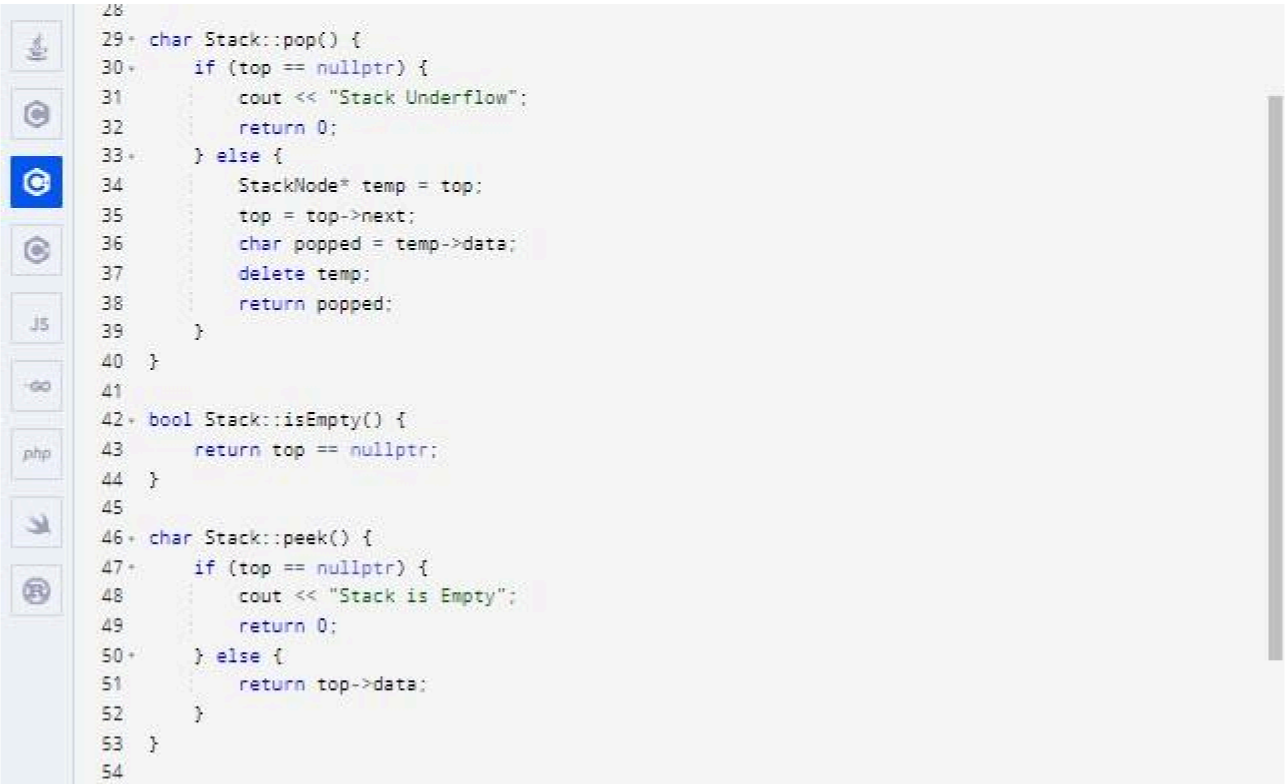
```
/tmp/lLKmxLLMwI.o
After the first PUSH, top of stack is: Top of Stack: 1
After the second PUSH, top of stack is: Top of Stack: 5
After the first POP operation, top of stack is: Top of Stack: 1
After the second POP operation, top of stack is: Stack is Empty.
Stack is Empty.


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

7. Supplementary Activity

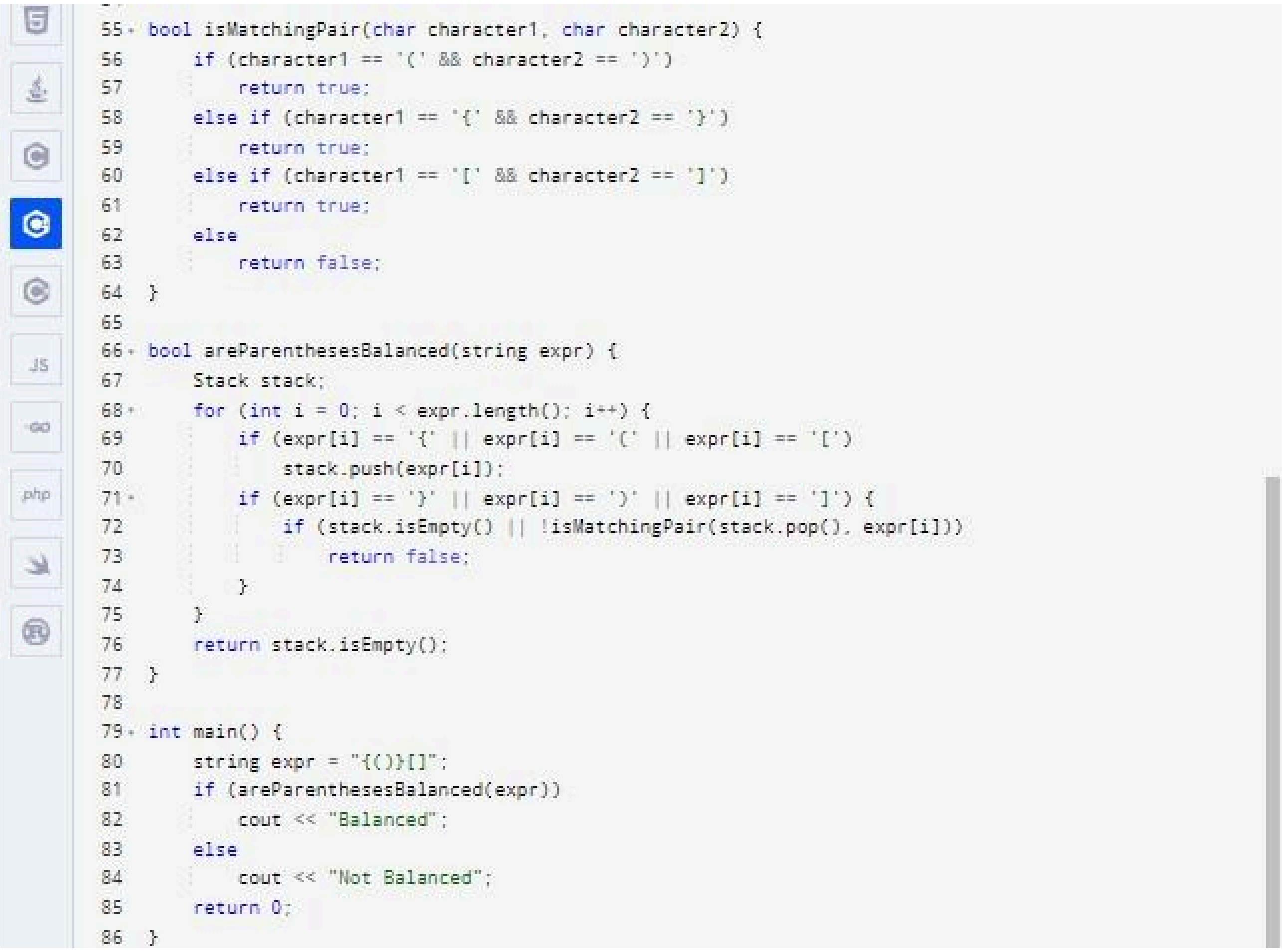
Stack using Arrays:



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

class StackNode {
public:
    char data;
    StackNode* next;
};

class Stack {
    StackNode* top;

public:
    Stack() { top = nullptr; }
    void push(char x);
    char pop();
    bool isEmpty();
    char peek();
};

void Stack::push(char x) {
    StackNode* newNode = new StackNode();
    newNode->data = x;
    newNode->next = top;
    top = newNode;
}
```

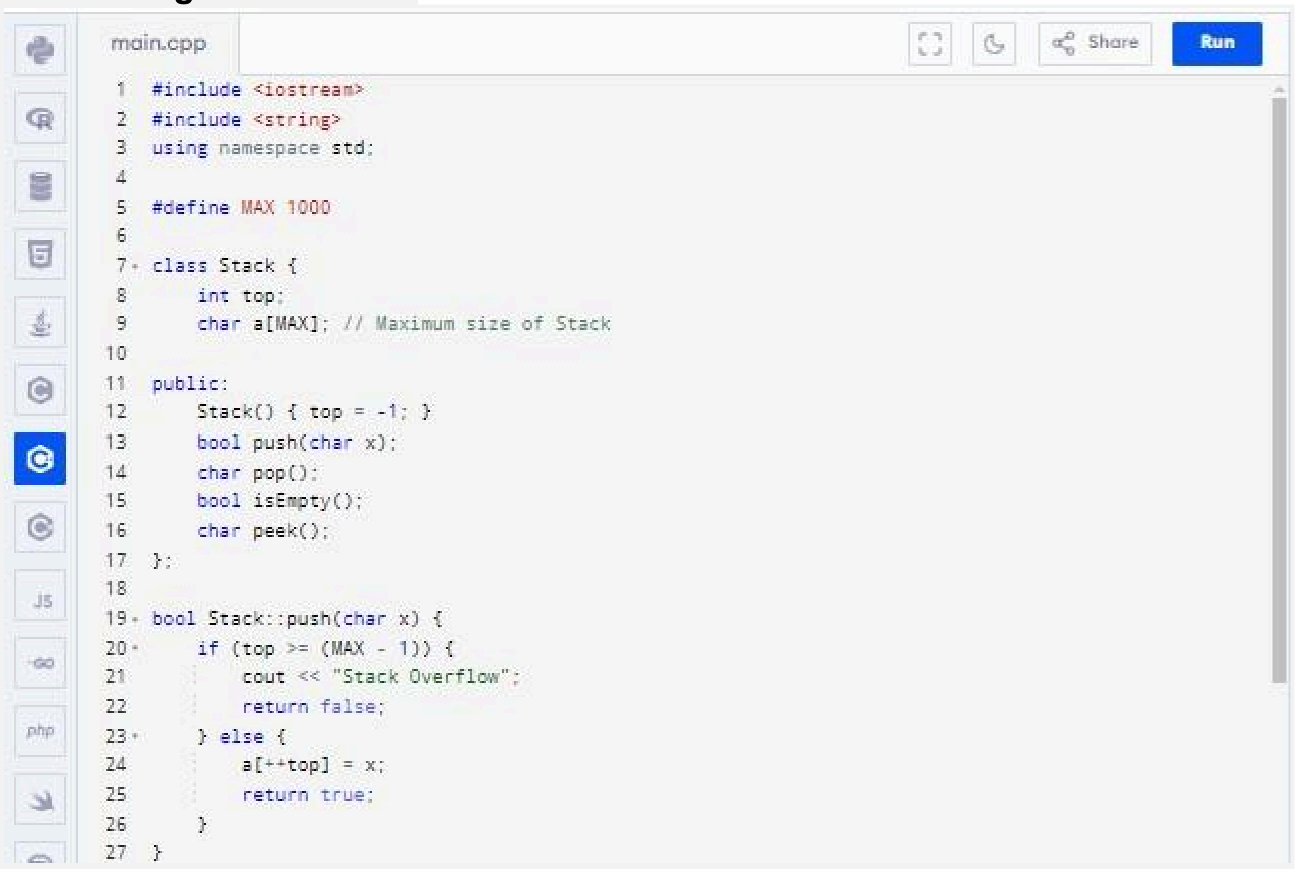


```cpp
char Stack::pop() {
    if (top == nullptr) {
        cout << "Stack Underflow";
        return 0;
    } else {
        StackNode* temp = top;
        top = top->next;
        char popped = temp->data;
        delete temp;
        return popped;
    }
}

bool Stack::isEmpty() {
    return top == nullptr;
}

char Stack::peek() {
    if (top == nullptr) {
        cout << "Stack is Empty";
        return 0;
    } else {
        return top->data;
    }
}

```

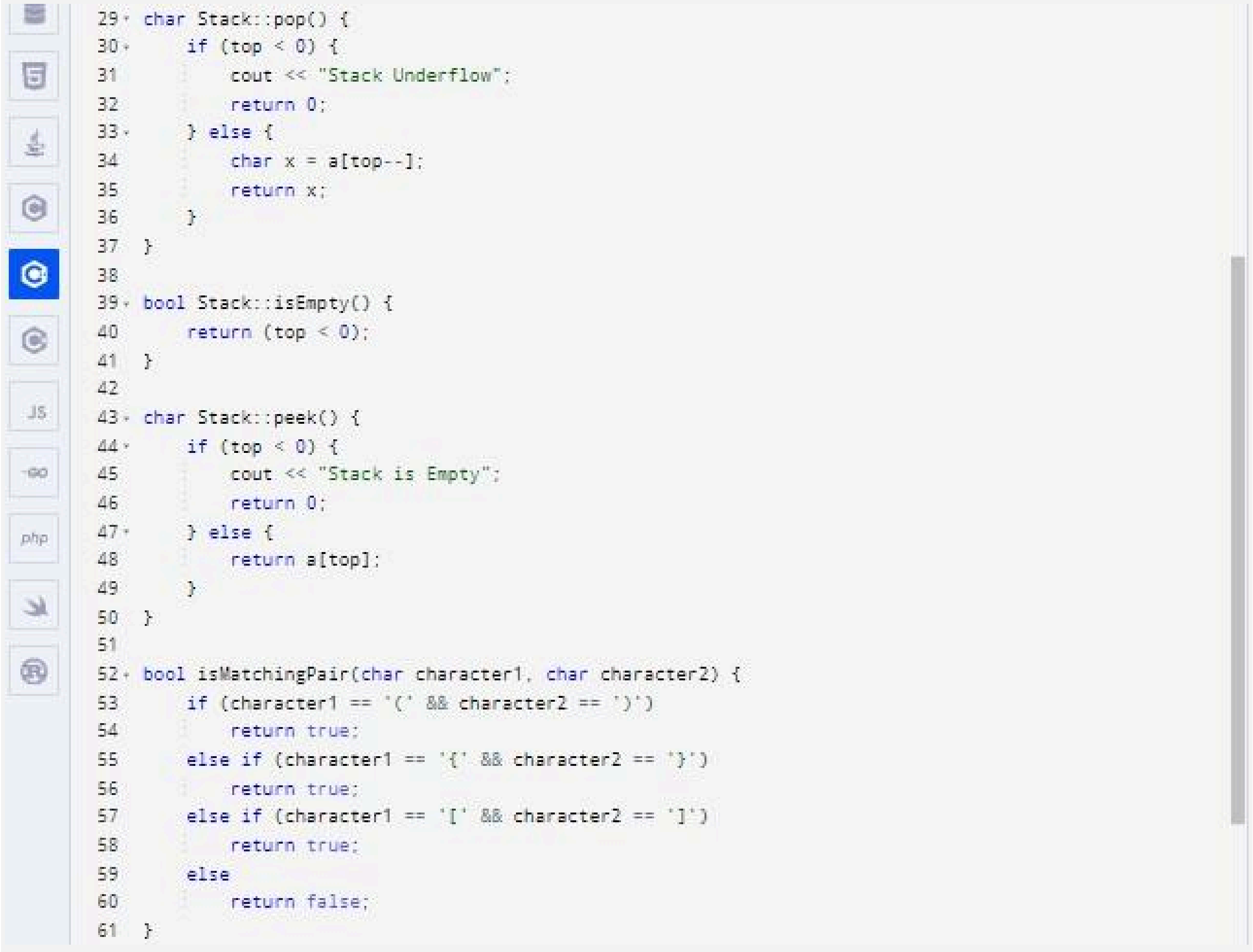


```cpp
bool isMatchingPair(char character1, char character2) {
    if (character1 == '(' && character2 == ')')
        return true;
    else if (character1 == '{' && character2 == '}')
        return true;
    else if (character1 == '[' && character2 == ']')
        return true;
    else
        return false;
}

bool areParenthesesBalanced(string expr) {
    Stack stack;
    for (int i = 0; i < expr.length(); i++) {
        if (expr[i] == '{' || expr[i] == '(' || expr[i] == '[')
            stack.push(expr[i]);
        if (expr[i] == '}' || expr[i] == ')' || expr[i] == ']') {
            if (stack.isEmpty() || !isMatchingPair(stack.pop(), expr[i]))
                return false;
        }
    }
    return stack.isEmpty();
}

int main() {
    string expr = "{()}[]";
    if (areParenthesesBalanced(expr))
        cout << "Balanced";
    else
        cout << "Not Balanced";
    return 0;
}
```

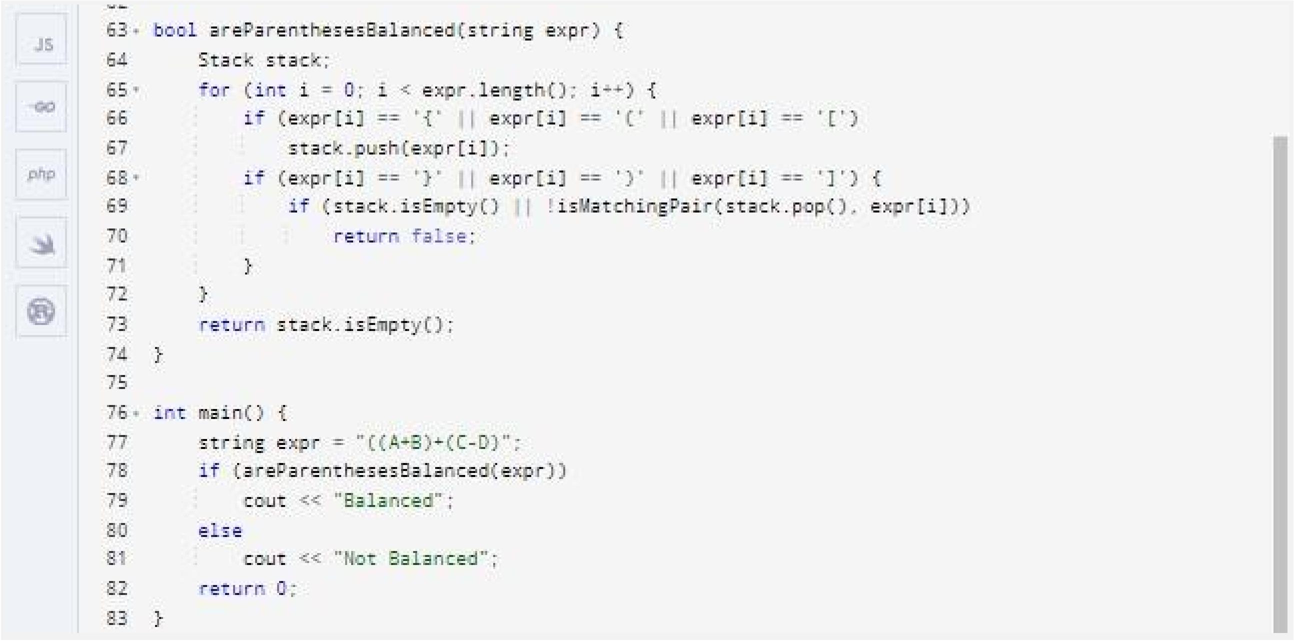
Stack using Linked Lists:



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

#define MAX 1000

class Stack {
    int top;
    char a[MAX]; // Maximum size of Stack

public:
    Stack() { top = -1; }
    bool push(char x);
    char pop();
    bool isEmpty();
    char peek();
};

bool Stack::push(char x) {
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    } else {
        a[++top] = x;
        return true;
    }
}
```

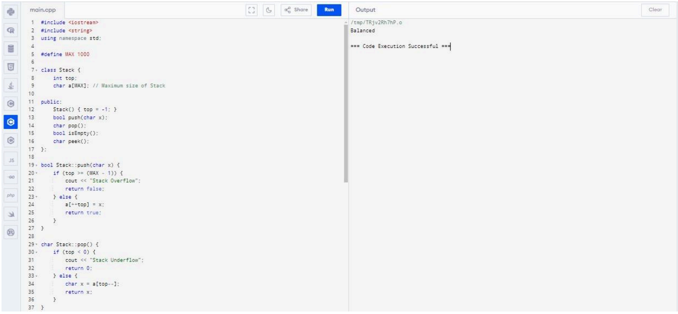


```cpp
char Stack::pop() {
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    } else {
        char x = a[top--];
        return x;
    }
}

bool Stack::isEmpty() {
    return (top < 0);
}

char Stack::peek() {
    if (top < 0) {
        cout << "Stack is Empty";
        return 0;
    } else {
        return a[top];
    }
}

bool isMatchingPair(char character1, char character2) {
    if (character1 == '(' && character2 == ')')
        return true;
    else if (character1 == '{' && character2 == '}')
        return true;
    else if (character1 == '[' && character2 == ']')
        return true;
    else
        return false;
}
```



```cpp
bool areParenthesesBalanced(string expr) {
    Stack stack;
    for (int i = 0; i < expr.length(); i++) {
        if (expr[i] == '{' || expr[i] == '(' || expr[i] == '[')
            stack.push(expr[i]);
        if (expr[i] == '}' || expr[i] == ')' || expr[i] == ']') {
            if (stack.isEmpty() || !isMatchingPair(stack.pop(), expr[i]))
                return false;
        }
    }
    return stack.isEmpty();
}

int main() {
    string expr = "((A+B)+(C-D)";
    if (areParenthesesBalanced(expr))
        cout << "Balanced";
    else
        cout << "Not Balanced";
    return 0;
}
```

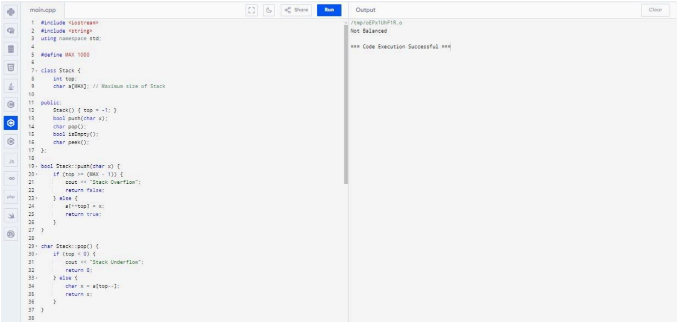
`(A+B)+(C-D)`



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

#define MAX 1000

class Stack {
    int top;
    char a[MAX]; // Maximum size of Stack

public:
    Stack() { top = -1; }
    bool push(char x);
    char pop();
    bool isEmpty();
    char peek();
};

bool Stack::push(char x) {
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    } else {
        a[++top] = x;
        return true;
    }
}

char Stack::pop() {
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    } else {
        char x = a[top--];
        return x;
    }
}
```

```
/tmp/TRjv2Rh7hP.o
Balanced

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

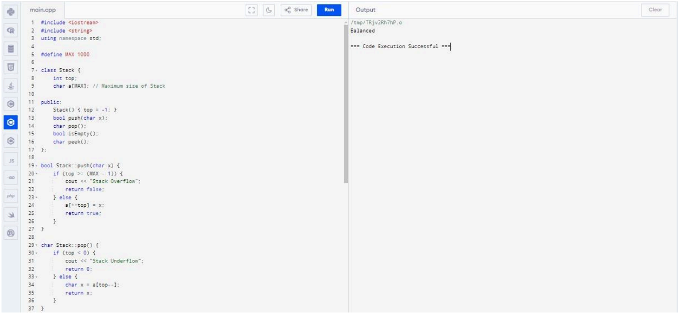
`((A+B)+(C-D)`



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

#define MAX 1000

class Stack {
    int top;
    char a[MAX]; // Maximum size of Stack

public:
    Stack() { top = -1; }
    bool push(char x);
    char pop();
    bool isEmpty();
    char peek();
};

bool Stack::push(char x) {
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    } else {
        a[++top] = x;
        return true;
    }
}

char Stack::pop() {
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    } else {
        char x = a[top--];
        return x;
    }
}

```

```
/tmp/oEPx1UhF1R.o
Not Balanced

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

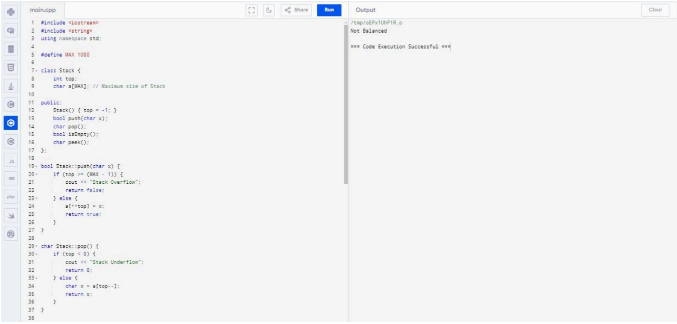
`((A+B)+[C-D])`



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

#define MAX 1000

class Stack {
    int top;
    char a[MAX]; // Maximum size of Stack

public:
    Stack() { top = -1; }
    bool push(char x);
    char pop();
    bool isEmpty();
    char peek();
};

bool Stack::push(char x) {
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    } else {
        a[++top] = x;
        return true;
    }
}

char Stack::pop() {
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    } else {
        char x = a[top--];
        return x;
    }
}
```

```
/tmp/TRjv2Rh7hP.o
Balanced

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

`((A+B]+[C-D]}`



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

#define MAX 1000

class Stack {
    int top;
    char a[MAX]; // Maximum size of Stack

public:
    Stack() { top = -1; }
    bool push(char x);
    char pop();
    bool isEmpty();
    char peek();
};

bool Stack::push(char x) {
    if (top >= (MAX - 1)) {
        cout << "Stack Overflow";
        return false;
    } else {
        a[++top] = x;
        return true;
    }
}

char Stack::pop() {
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    } else {
        char x = a[top--];
        return x;
    }
}

```

```
/tmp/oEPx1UhF1R.o
Not Balanced

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

8. Conclusion

Implementing a stack in C++ has provided a solid grasp of this fundamental data structure, which operates on a Last In, First Out (LIFO) basis. You've learned key operations like push, pop, and peek, which are essential for managing data efficiently. This experience not only enhances your problem-solving skills but also prepares you for more complex data structures and algorithms. Mastering stack implementation is a significant milestone in your programming journey, equipping you to handle various computational tasks effectively.

9. Assessment Rubric