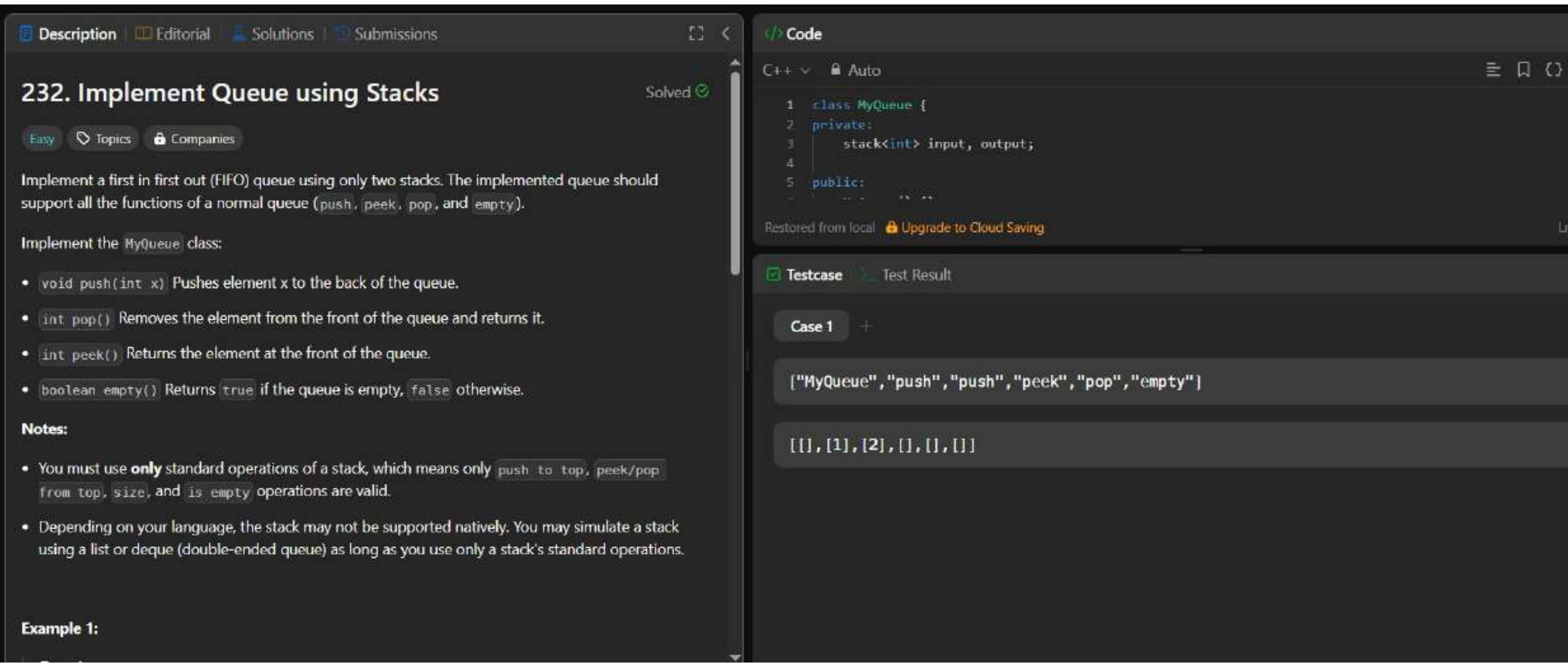


232. Implement Queue using Stacks

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.

Notes:

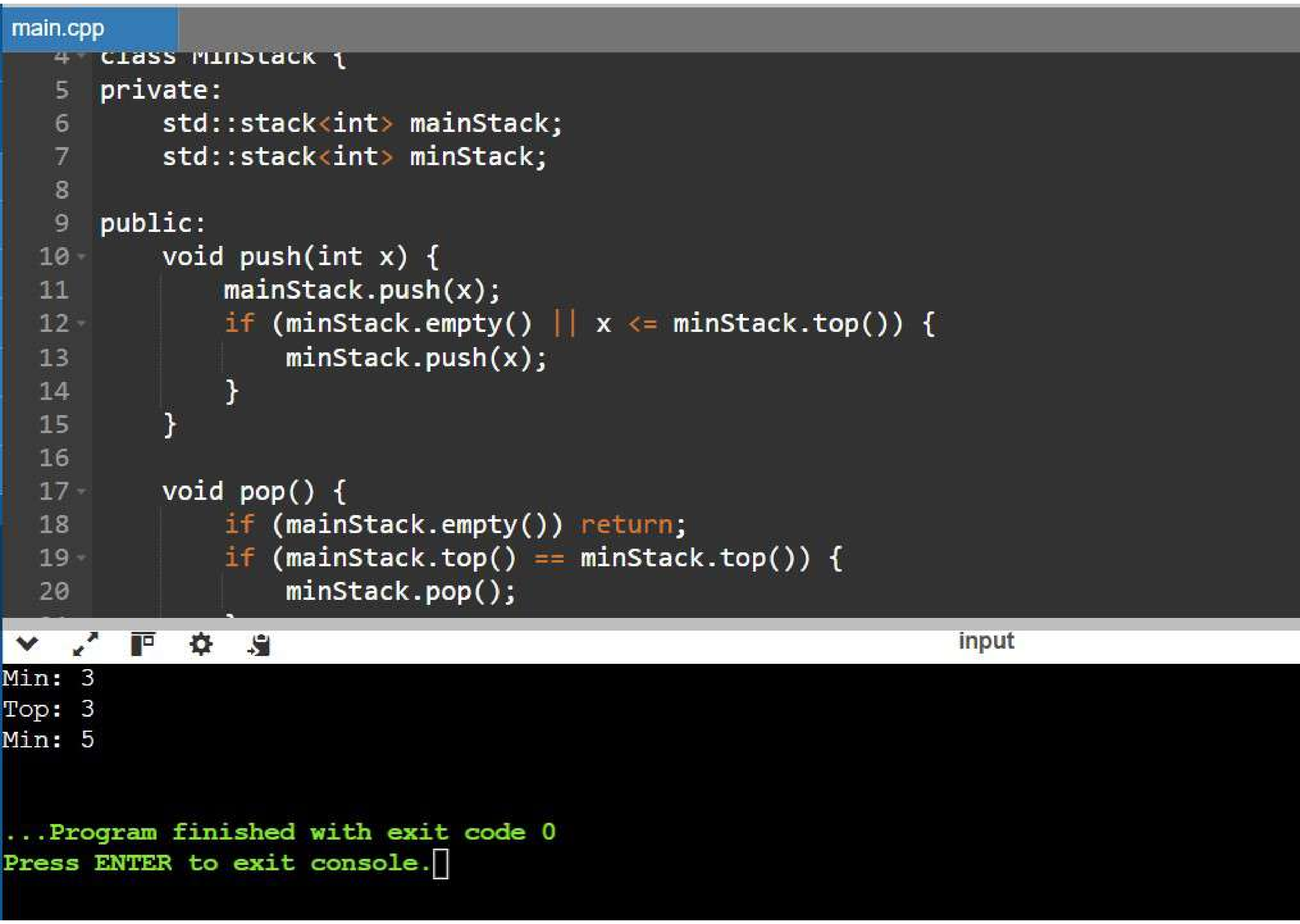
- You must use **only** standard operations of a stack, which means only `push to top`, `peek/pop from top`, `size`, and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:

```cpp
class MyQueue {
private:
    stack<int> input, output;

public:
```

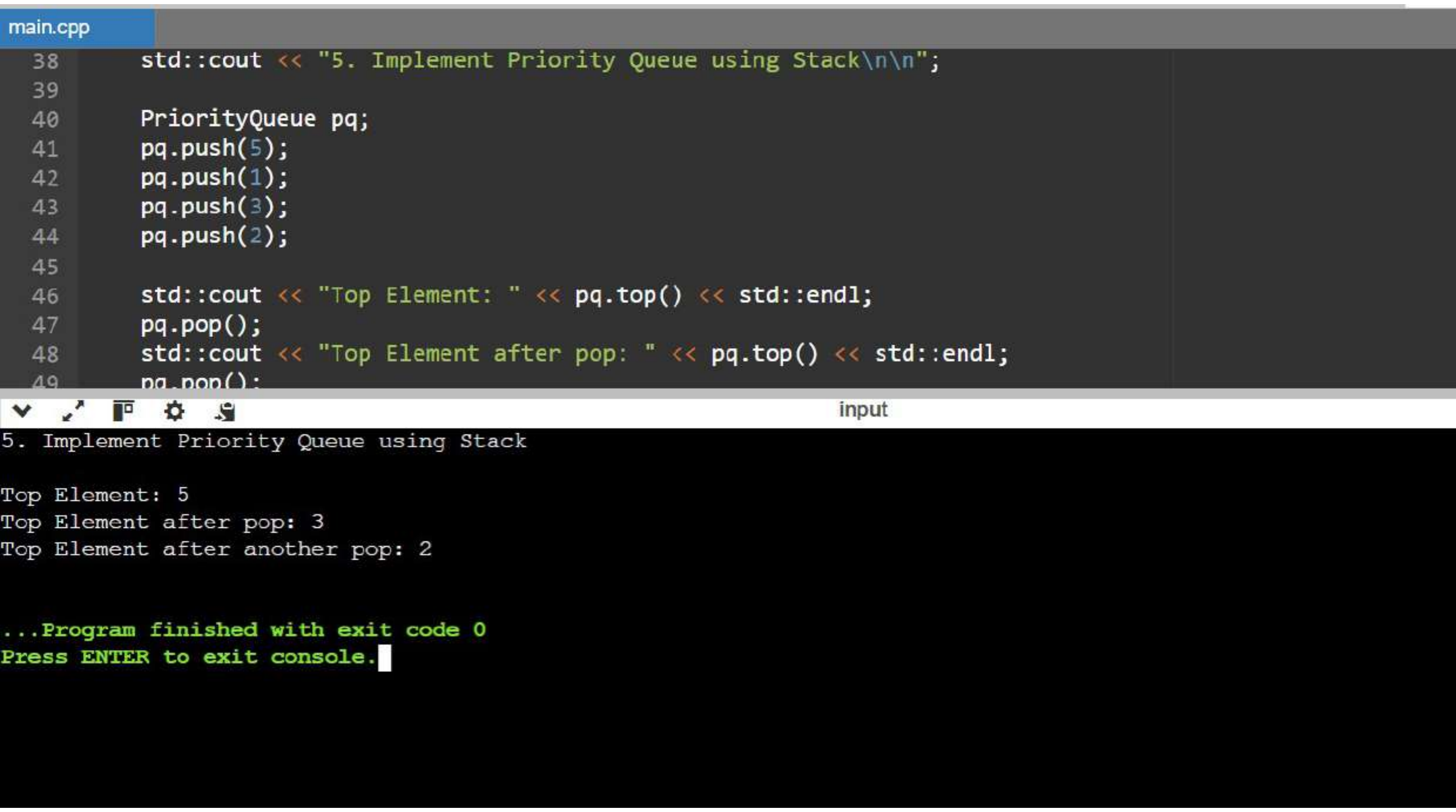
```
["MyQueue","push","push","peek","pop","empty"]
```

```
[[],[1],[2],[],[],[]]
```



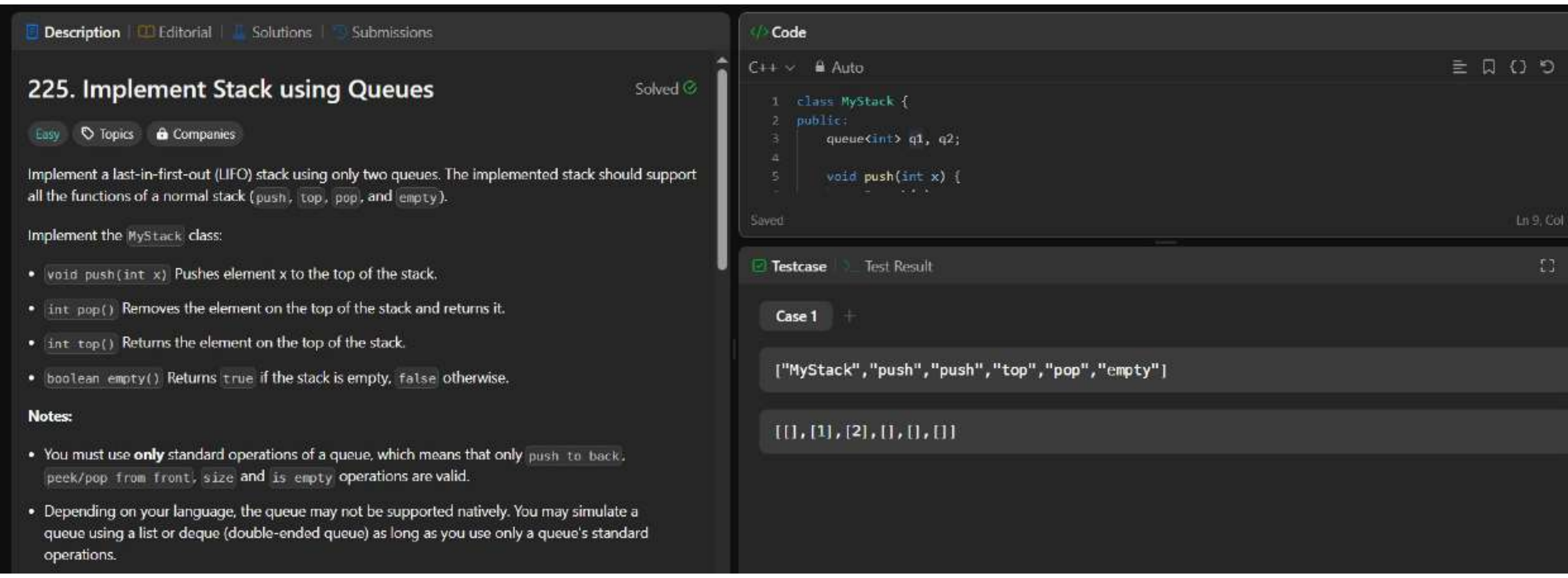
```cpp
class MinStack {
private:
    std::stack<int> mainStack;
    std::stack<int> minStack;

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

    void pop() {
        if (mainStack.empty()) return;
        if (mainStack.top() == minStack.top()) {
            minStack.pop();
        }
```

```
Min: 3
Top: 3
Min: 5


...Program finished with exit code 0
Press ENTER to exit console.
```



```cpp
    std::cout << "5. Implement Priority Queue using Stack\n\n";

    PriorityQueue pq;
    pq.push(5);
    pq.push(1);
    pq.push(3);
    pq.push(2);

    std::cout << "Top Element: " << pq.top() << std::endl;
    pq.pop();
    std::cout << "Top Element after pop: " << pq.top() << std::endl;
    pq.pop();
```

```
5. Implement Priority Queue using Stack

Top Element: 5
Top Element after pop: 3
Top Element after another pop: 2


...Program finished with exit code 0
Press ENTER to exit console.
```



225. Implement Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

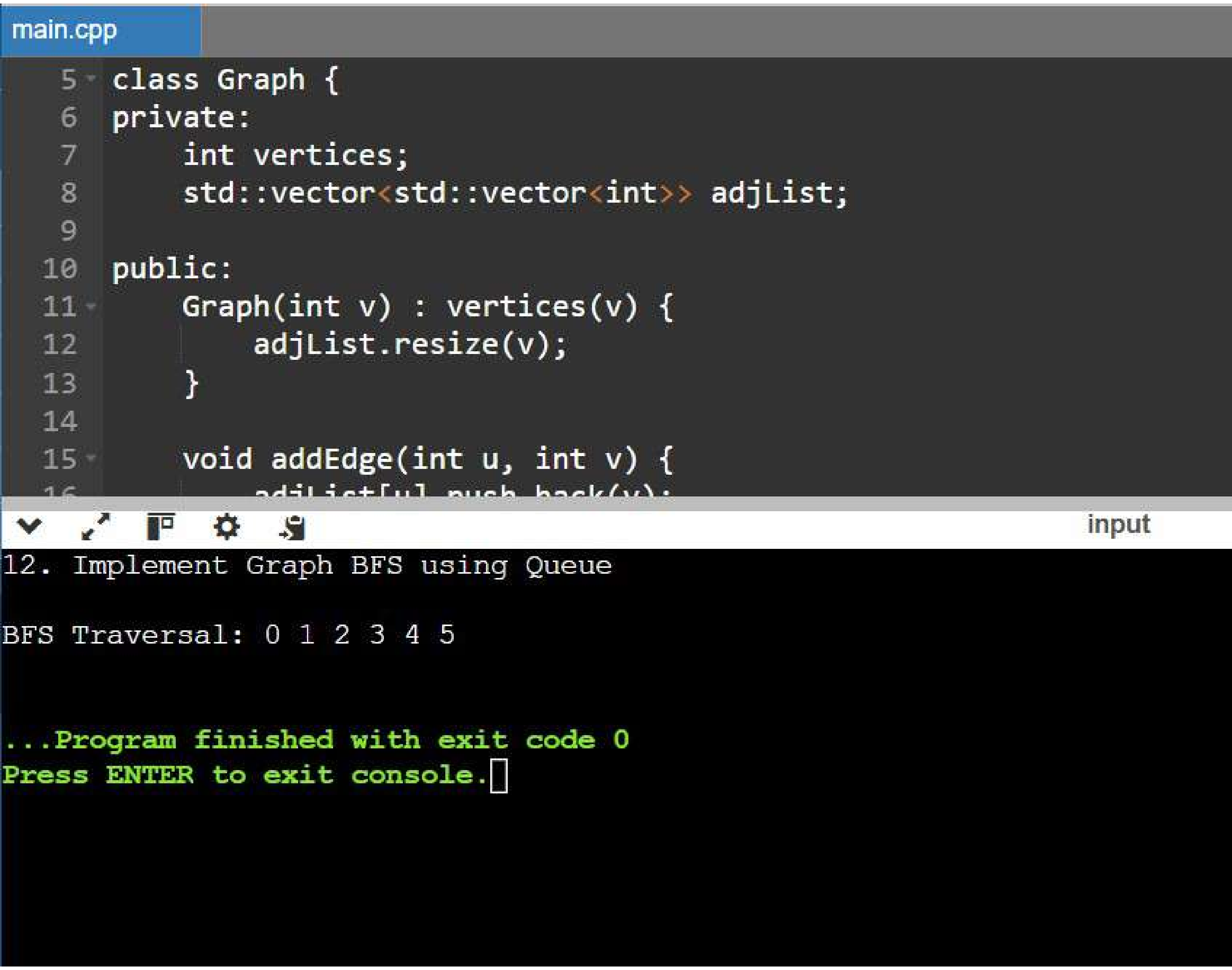
Notes:

- You must use **only** standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

```
class MyStack {
public:
    queue<int> q1, q2;

    void push(int x) {
```

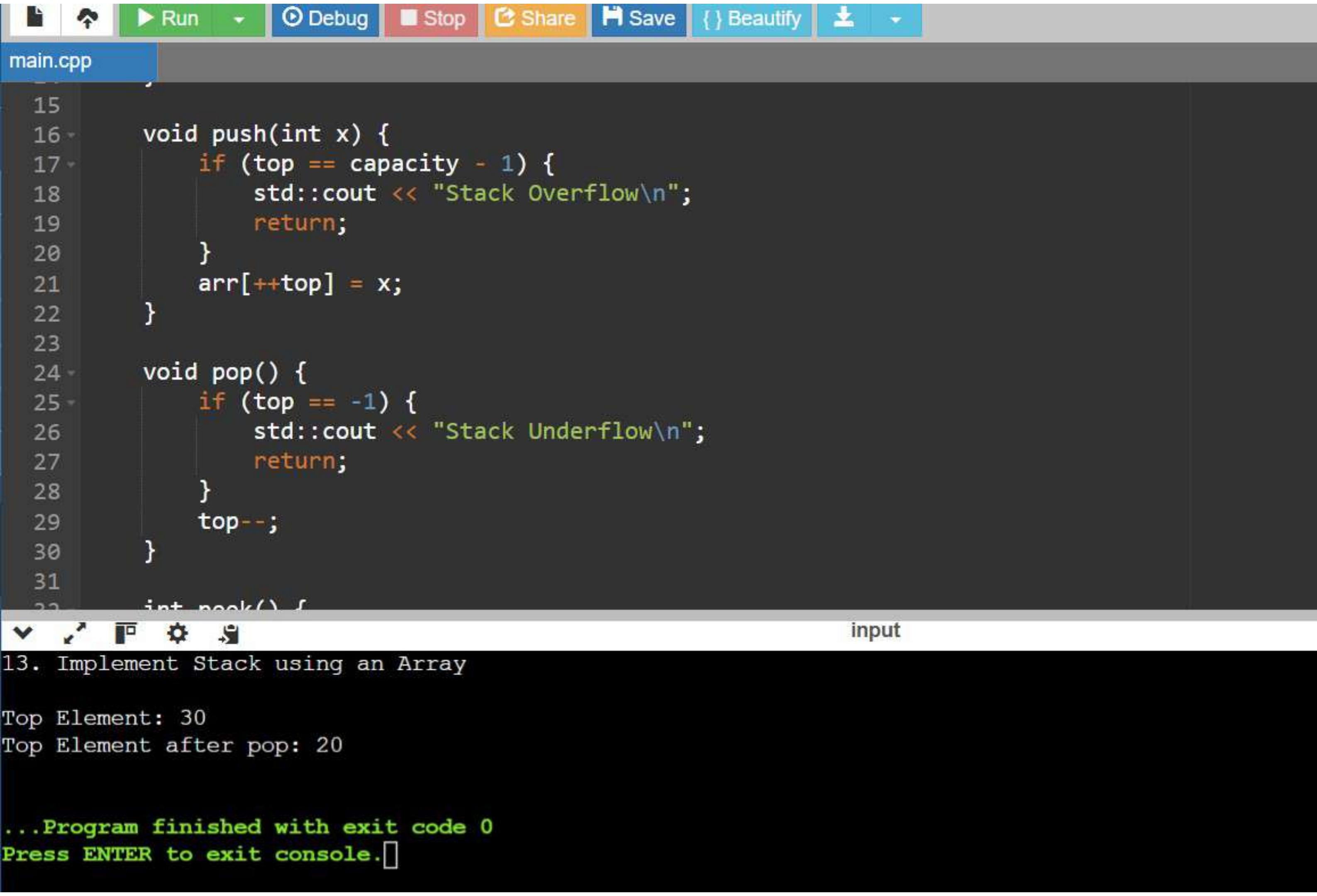
```
["MyStack","push","push","top","pop","empty"]
```

```
[[],[1],[2],[],[],[]]
```



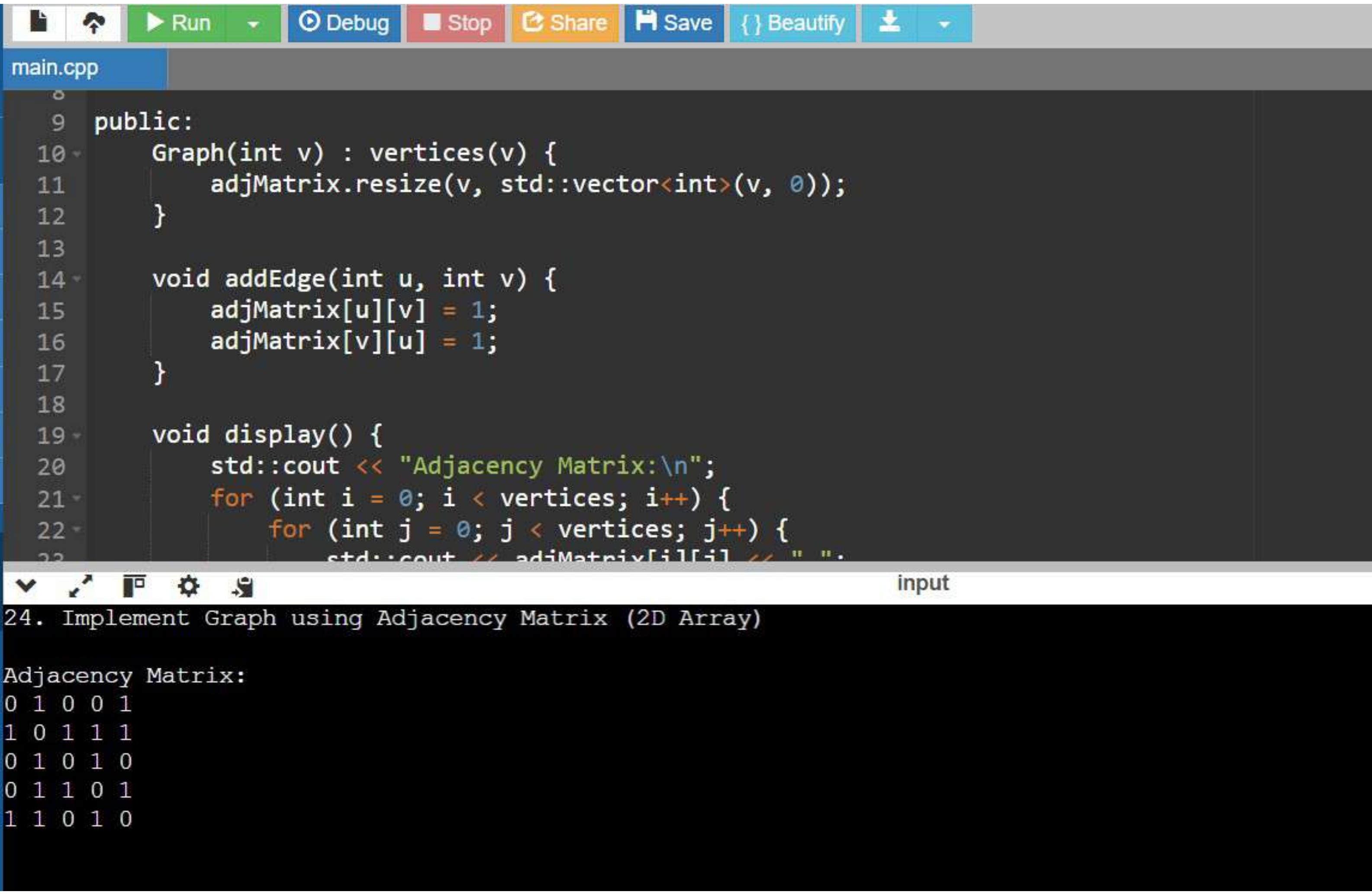
```cpp
class Graph {
private:
    int vertices;
    std::vector<std::vector<int>> adjList;

public:
    Graph(int v) : vertices(v) {
        adjList.resize(v);
    }

    void addEdge(int u, int v) {
        adjList[u].push_back(v);
```

```
12. Implement Graph BFS using Queue

BFS Traversal: 0 1 2 3 4 5


...Program finished with exit code 0
Press ENTER to exit console.
```



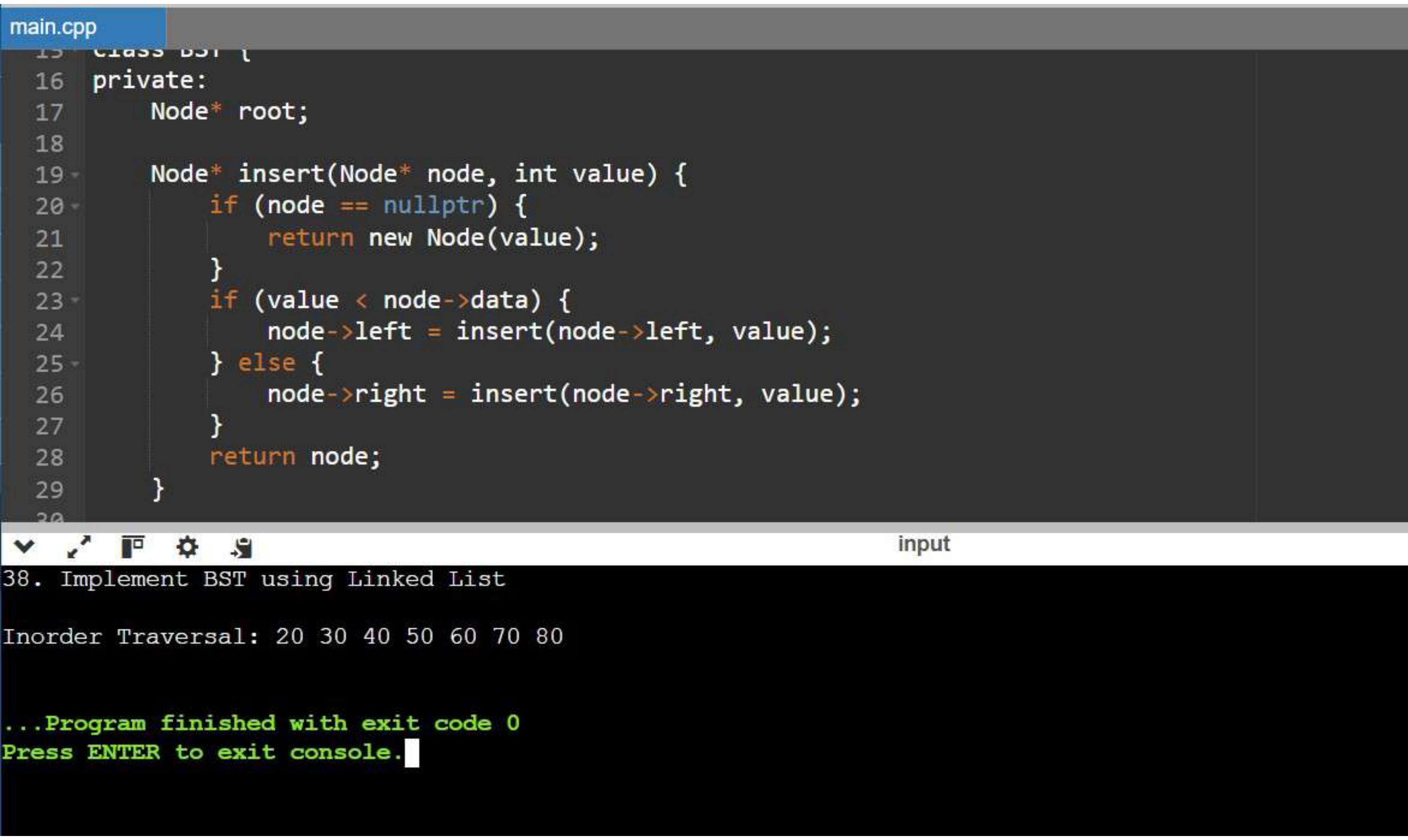
```cpp

    void push(int x) {
        if (top == capacity - 1) {
            std::cout << "Stack Overflow\n";
            return;
        }
        arr[++top] = x;
    }

    void pop() {
        if (top == -1) {
            std::cout << "Stack Underflow\n";
            return;
        }
        top--;
    }

    int peek() {
```

```
13. Implement Stack using an Array

Top Element: 30
Top Element after pop: 20


...Program finished with exit code 0
Press ENTER to exit console.
```



```cpp
public:
    Graph(int v) : vertices(v) {
        adjMatrix.resize(v, std::vector<int>(v, 0));
    }

    void addEdge(int u, int v) {
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
    }

    void display() {
        std::cout << "Adjacency Matrix:\n";
        for (int i = 0; i < vertices; i++) {
            for (int j = 0; j < vertices; j++) {
                std::cout << adjMatrix[i][j] << " ";
```

```
24. Implement Graph using Adjacency Matrix (2D Array)

Adjacency Matrix:
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 1 0 1
1 1 0 1 0
```



```cpp
class BST {
private:
    Node* root;

    Node* insert(Node* node, int value) {
        if (node == nullptr) {
            return new Node(value);
        }
        if (value < node->data) {
            node->left = insert(node->left, value);
        } else {
            node->right = insert(node->right, value);
        }
        return node;
    }
```

```
38. Implement BST using Linked List

Inorder Traversal: 20 30 40 50 60 70 80


...Program finished with exit code 0
Press ENTER to exit console.
```