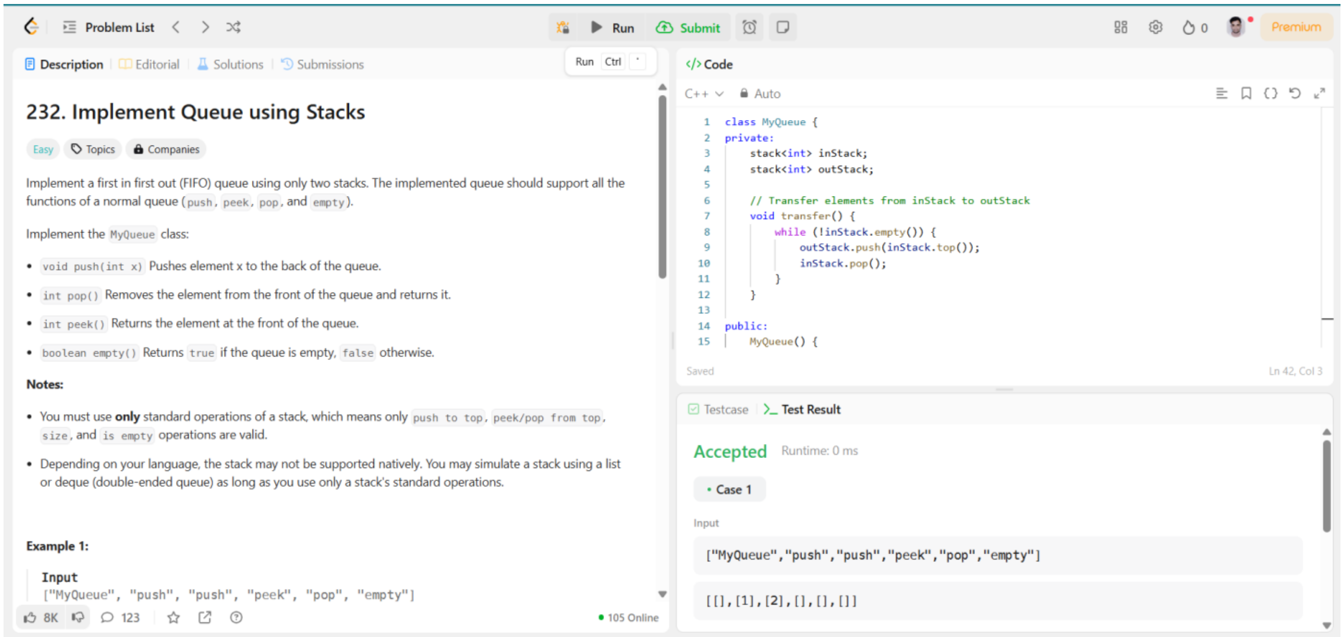
DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 6

Student Name: Saurav Ashiwal **UID: 22BCS13250**
Branch: CSE **Section/Group: 614/B**
Semester: 6 **Date of Performance: 12/03/2025**
Subject Name: AP lab-2 **Subject Code: 22CSP-351**

Q 1. Implement Queue using Stack



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:

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

```cpp
class MyQueue {
private:
    stack<int> inStack;
    stack<int> outStack;

    // Transfer elements from inStack to outStack
    void transfer() {
        while (!inStack.empty()) {
            outStack.push(inStack.top());
            inStack.pop();
        }
    }

public:
    MyQueue() {
```

Accepted Runtime: 0 ms

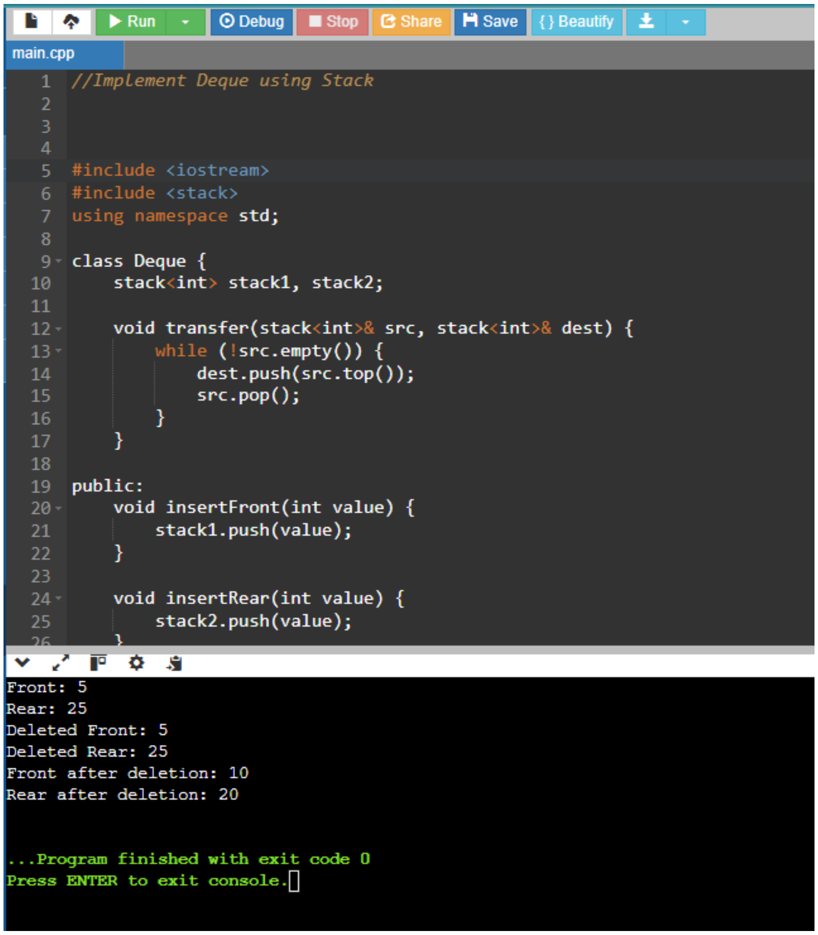
- Case 1

Input

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

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

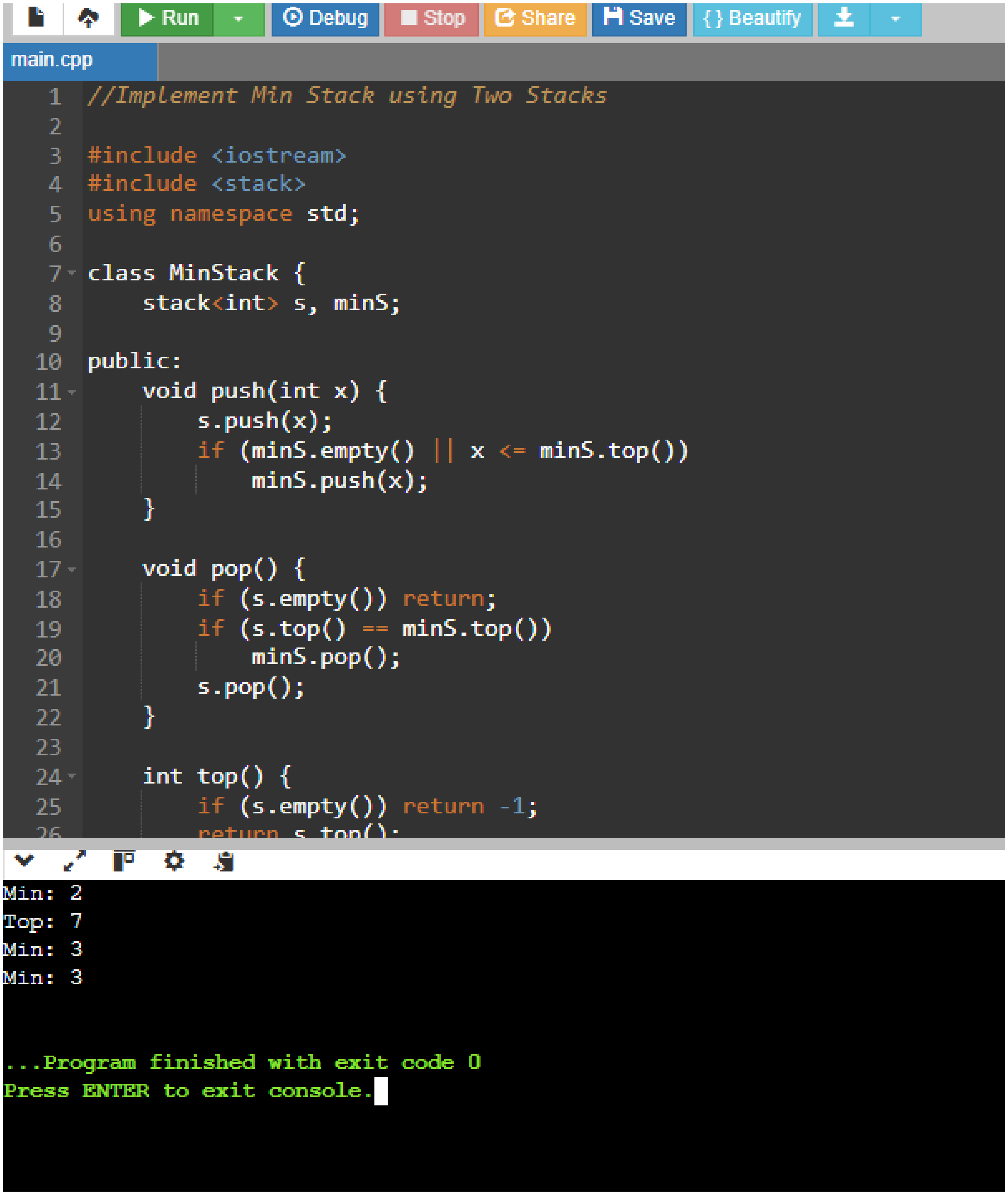
2.Implement Deque using Stack



```cpp
//Implement Deque using Stack



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

class Deque {
    stack<int> stack1, stack2;

    void transfer(stack<int>& src, stack<int>& dest) {
        while (!src.empty()) {
            dest.push(src.top());
            src.pop();
        }
    }

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

    void insertRear(int value) {
        stack2.push(value);
    }
```

```
Front: 5
Rear: 25
Deleted Front: 5
Deleted Rear: 25
Front after deletion: 10
Rear after deletion: 20


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

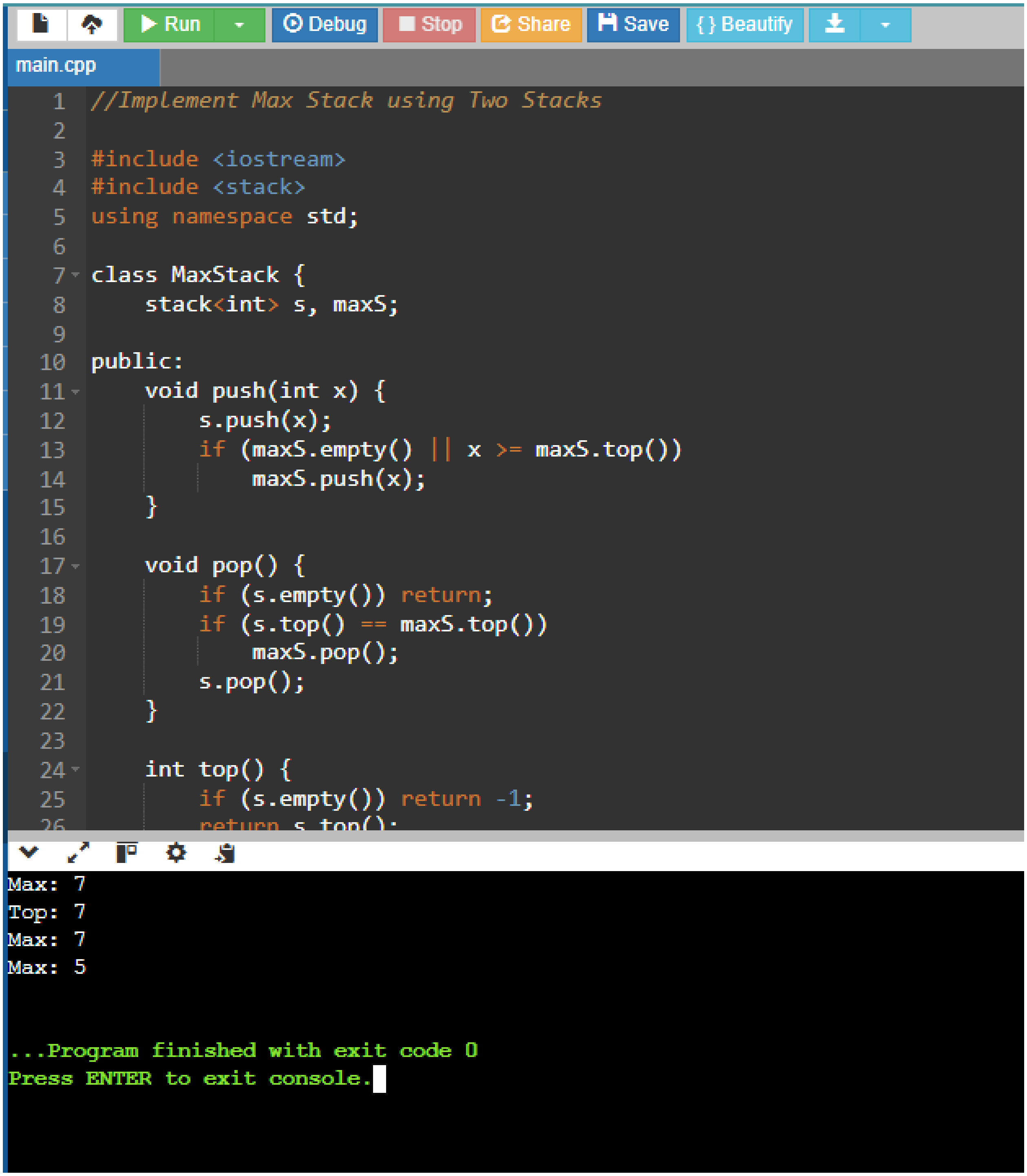
3.Implement Min Stack using Two Stacks



```cpp
//Implement Min Stack using Two Stacks

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

class MinStack {
    stack<int> s, minS;

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

    void pop() {
        if (s.empty()) return;
        if (s.top() == minS.top())
            minS.pop();
        s.pop();
    }

    int top() {
        if (s.empty()) return -1;
        return s.top();
```

```
Min: 2
Top: 7
Min: 3
Min: 3


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

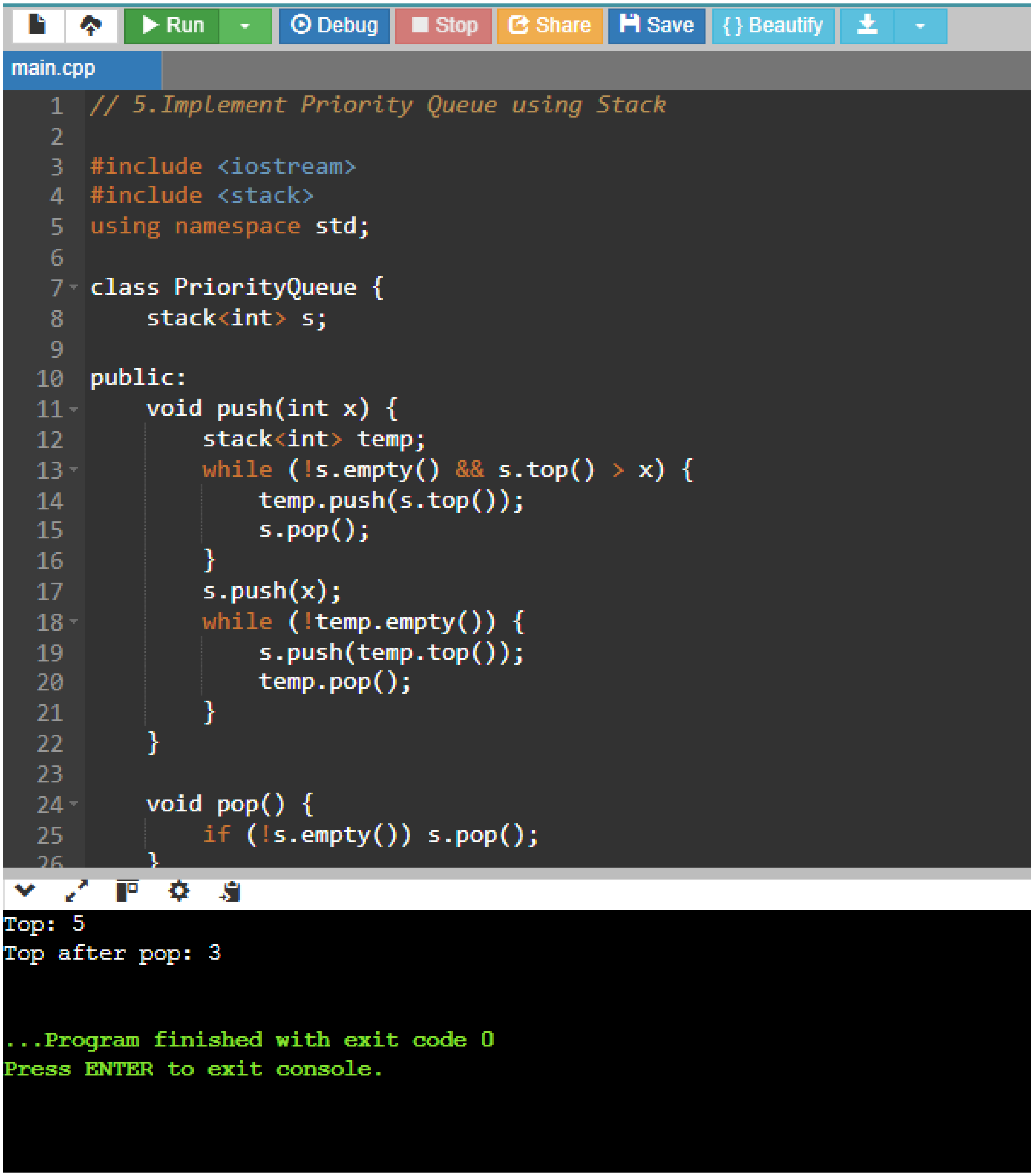
4. Implement Max Stack using Two Stacks



```cpp
//Implement Max Stack using Two Stacks

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

class MaxStack {
    stack<int> s, maxS;

public:
    void push(int x) {
        s.push(x);
        if (maxS.empty() || x >= maxS.top())
            maxS.push(x);
    }

    void pop() {
        if (s.empty()) return;
        if (s.top() == maxS.top())
            maxS.pop();
        s.pop();
    }

    int top() {
        if (s.empty()) return -1;
        return s.top();
```

```
Max: 7
Top: 7
Max: 7
Max: 5


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

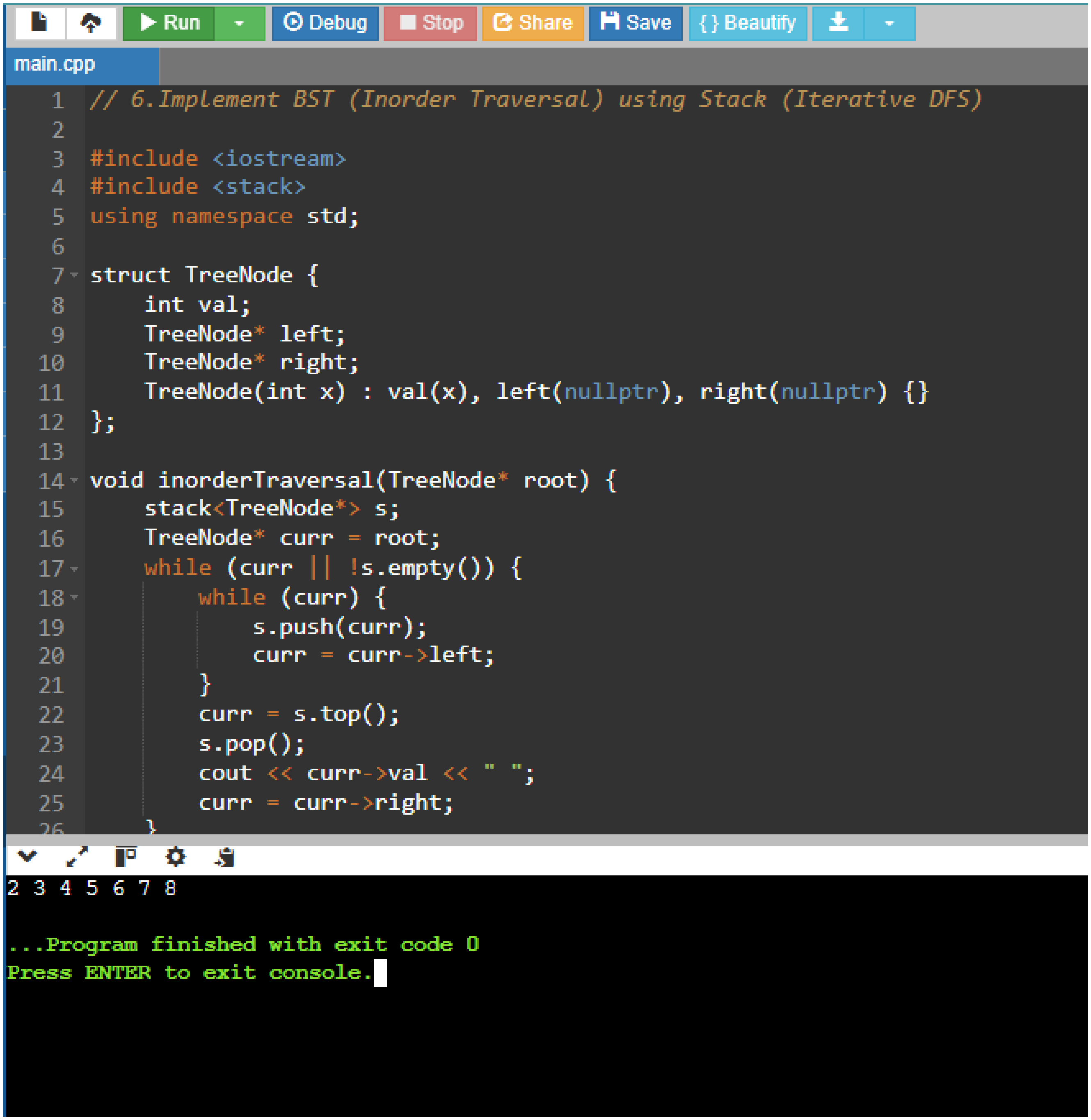
5. Implement Priority Queue using Stack



```cpp
// 5.Implement Priority Queue using Stack

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

class PriorityQueue {
    stack<int> s;

public:
    void push(int x) {
        stack<int> temp;
        while (!s.empty() && s.top() > x) {
            temp.push(s.top());
            s.pop();
        }
        s.push(x);
        while (!temp.empty()) {
            s.push(temp.top());
            temp.pop();
        }
    }

    void pop() {
        if (!s.empty()) s.pop();
    }
```

```
Top: 5
Top after pop: 3


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

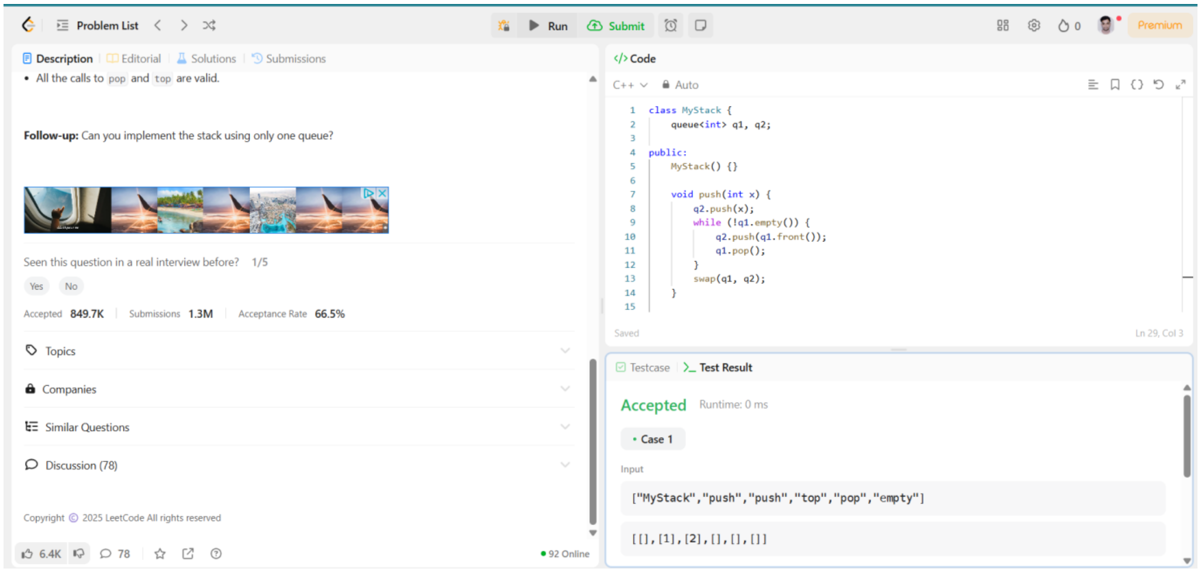
6. Implement BST (Inorder Traversal) using Stack (Iterative DFS)



```cpp
// 6.Implement BST (Inorder Traversal) using Stack (Iterative DFS)

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

void inorderTraversal(TreeNode* root) {
    stack<TreeNode*> s;
    TreeNode* curr = root;
    while (curr || !s.empty()) {
        while (curr) {
            s.push(curr);
            curr = curr->left;
        }
        curr = s.top();
        s.pop();
        cout << curr->val << " ";
        curr = curr->right;
    }
```

```
2 3 4 5 6 7 8

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

8. Implement Stack using Queue



- All the calls to `pop` and `top` are valid.

Follow-up: Can you implement the stack using only one queue?

Seen this question in a real interview before? 1/5

Accepted 849.7K | Submissions 1.3M | Acceptance Rate 66.5%

Copyright © 2025 LeetCode All rights reserved

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

public:
    MyStack() {}

    void push(int x) {
        q2.push(x);
        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }
        swap(q1, q2);
    }

```

Saved   Ln 29, Col 3

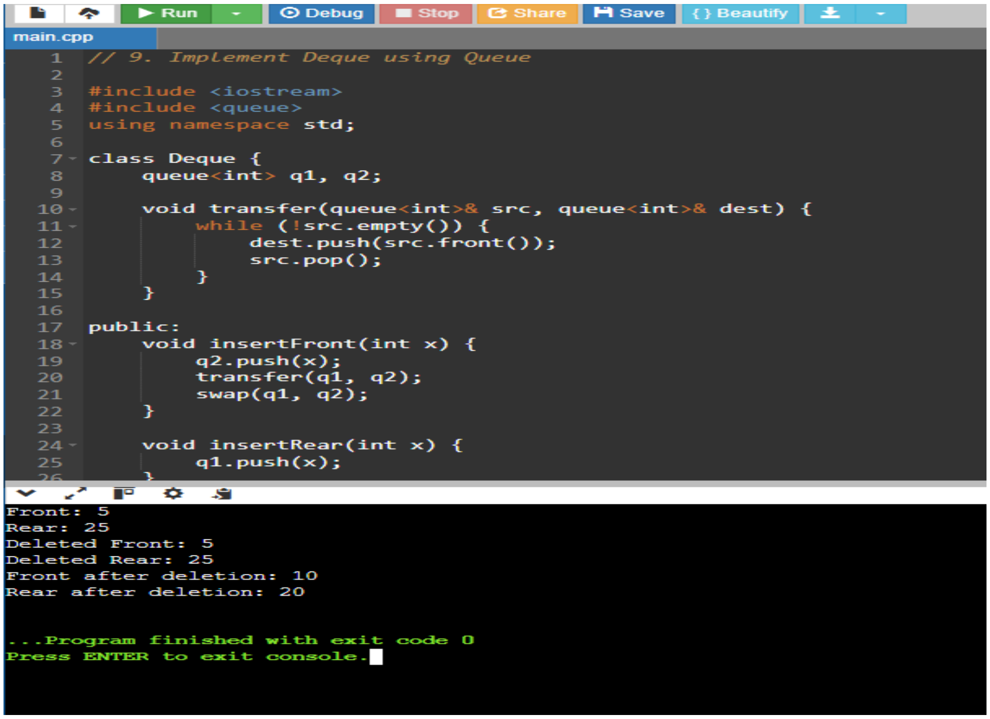
Accepted Runtime: 0 ms

Case 1

Input

```
["MyStack","push","push","top","pop","empty"]
[[],[1],[2],[],[],[]]
```

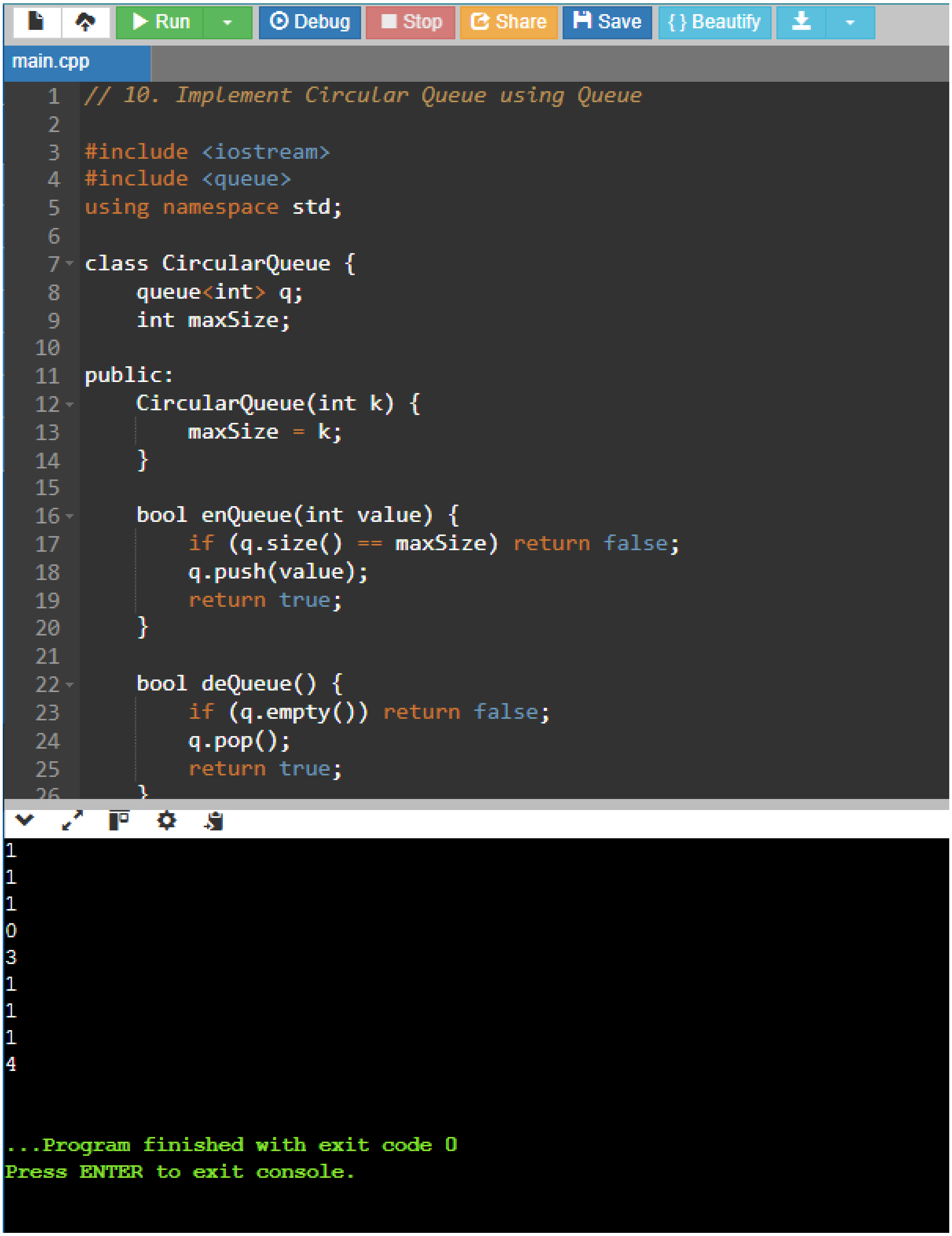
9. Implement Deque using Queue



```cpp
// 9. Implement Deque using Queue

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

class Deque {
    queue<int> q1, q2;

    void transfer(queue<int>& src, queue<int>& dest) {
        while (!src.empty()) {
            dest.push(src.front());
            src.pop();
        }
    }

public:
    void insertFront(int x) {
        q2.push(x);
        transfer(q1, q2);
        swap(q1, q2);
    }

    void insertRear(int x) {
        q1.push(x);
    }
```

```
Front: 5
Rear: 25
Deleted Front: 5
Deleted Rear: 25
Front after deletion: 10
Rear after deletion: 20


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

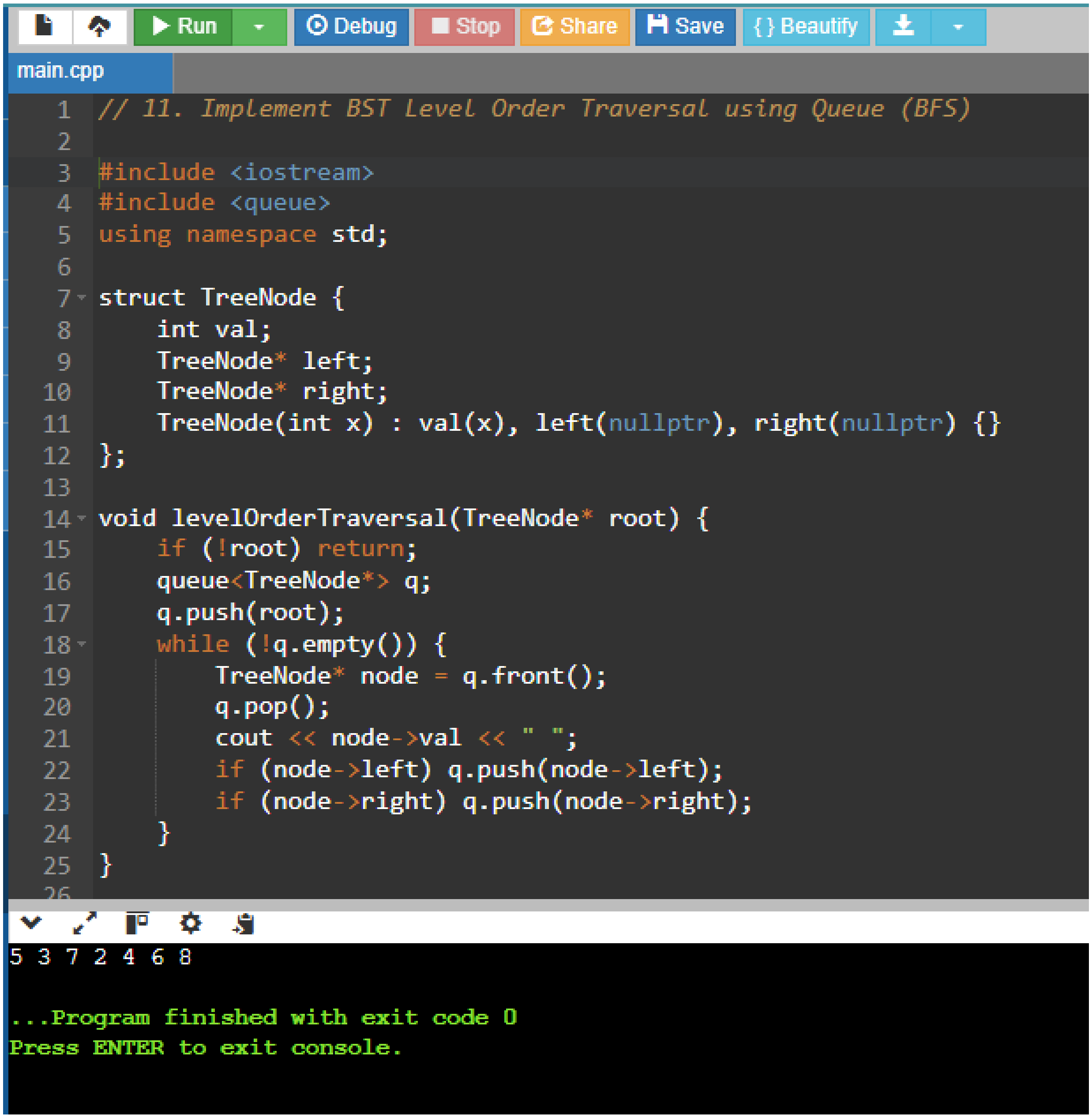
10. Implement Circular Queue using Queue



```cpp
// 10. Implement Circular Queue using Queue

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

class CircularQueue {
    queue<int> q;
    int maxSize;

public:
    CircularQueue(int k) {
        maxSize = k;
    }

    bool enQueue(int value) {
        if (q.size() == maxSize) return false;
        q.push(value);
        return true;
    }

    bool deQueue() {
        if (q.empty()) return false;
        q.pop();
        return true;
    }
```

```
1
1
1
0
3
1
1
1
4


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

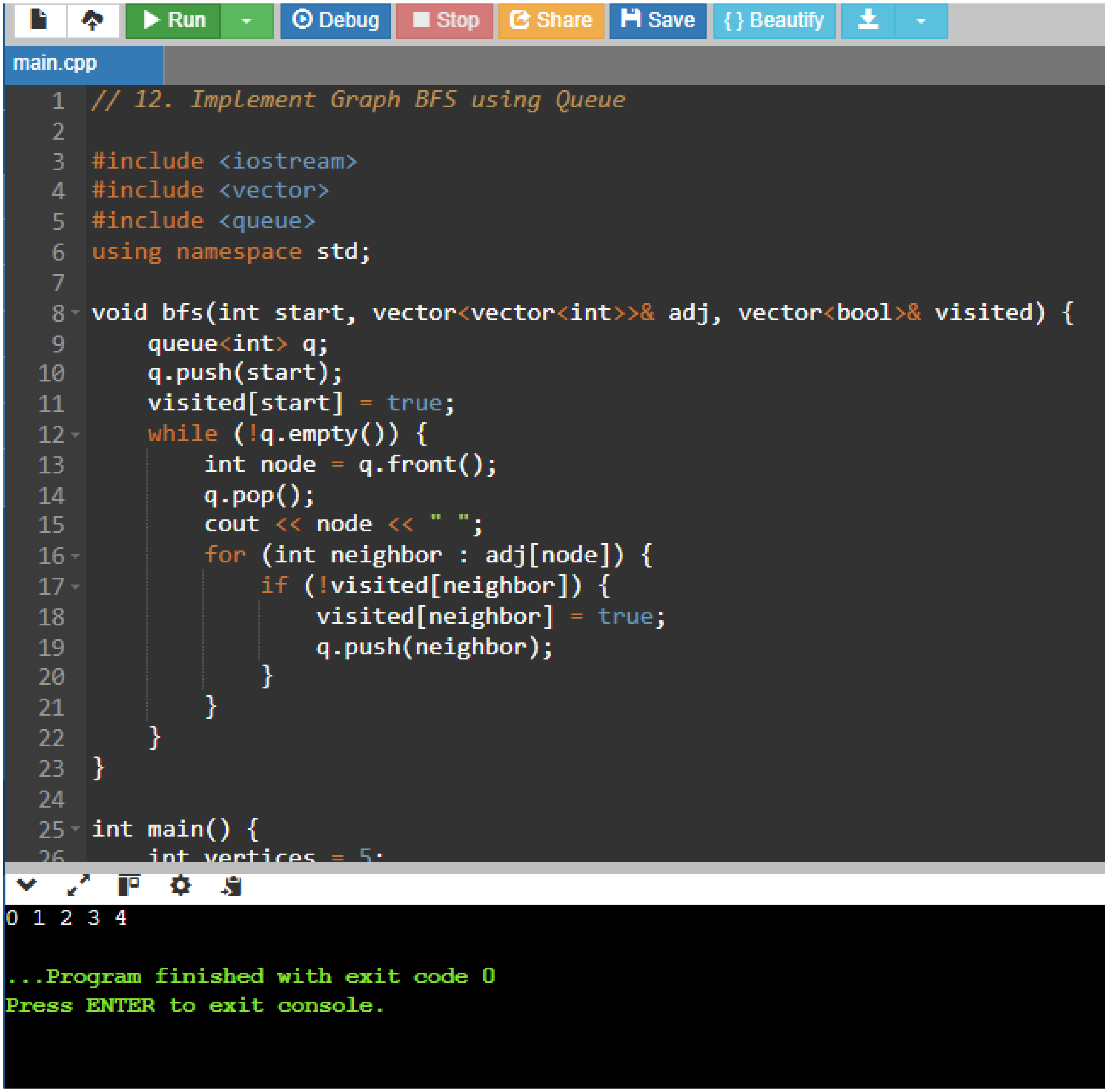
11. Implement BST Level Order Traversal using Queue (BFS)



```cpp
// 11. Implement BST Level Order Traversal using Queue (BFS)

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

void levelOrderTraversal(TreeNode* root) {
    if (!root) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();
        cout << node->val << " ";
        if (node->left) q.push(node->left);
        if (node->right) q.push(node->right);
    }
}
```

```
5 3 7 2 4 6 8

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

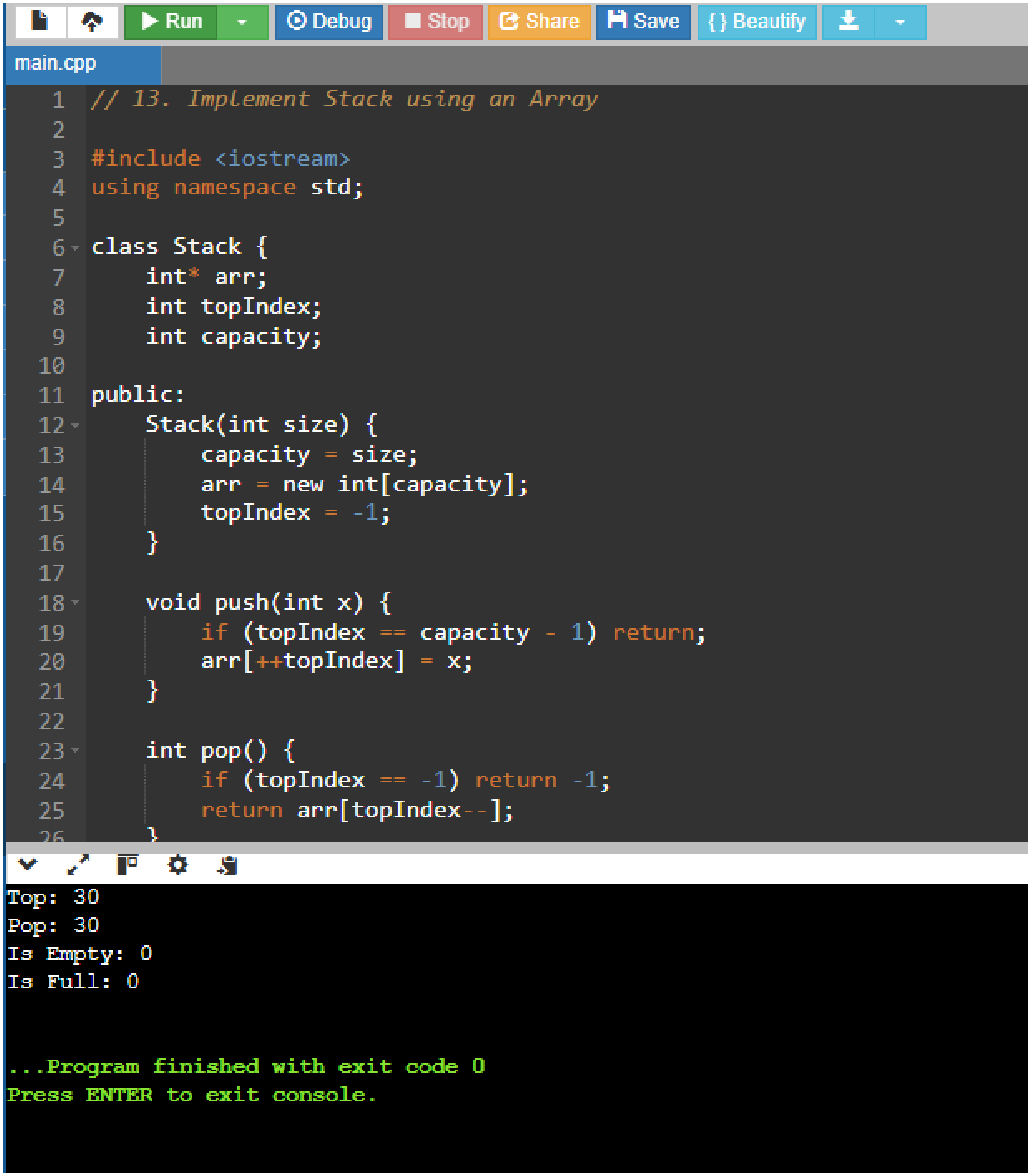
12. Implement Graph BFS using Queue



```cpp
// 12. Implement Graph BFS using Queue

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

void bfs(int start, vector<vector<int>>& adj, vector<bool>& visited) {
    queue<int> q;
    q.push(start);
    visited[start] = true;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
}

int main() {
    int vertices = 5;
```

```
0 1 2 3 4

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

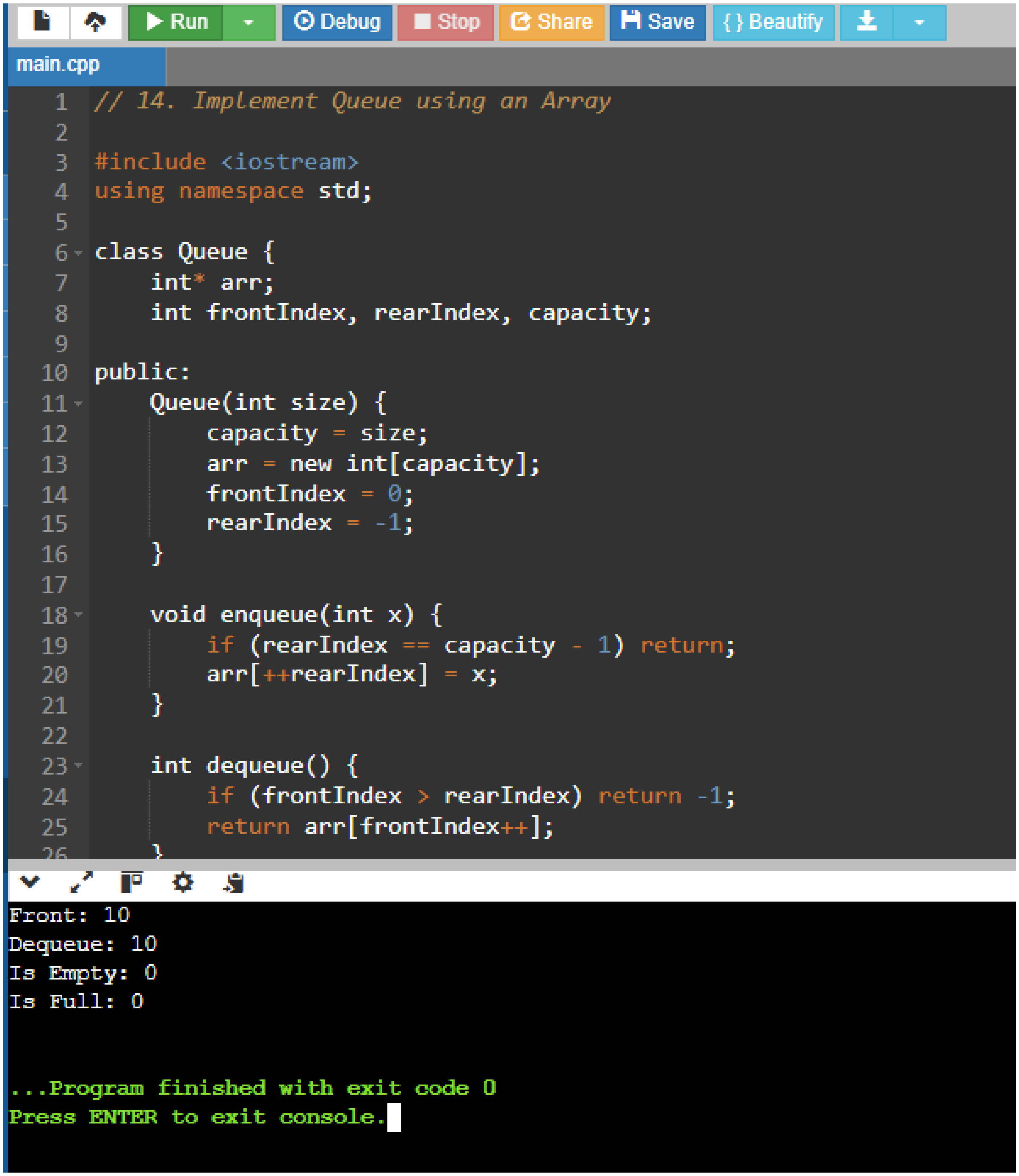
13. Implement Stack using an Array



```cpp
// 13. Implement Stack using an Array

#include <iostream>
using namespace std;

class Stack {
    int* arr;
    int topIndex;
    int capacity;

public:
    Stack(int size) {
        capacity = size;
        arr = new int[capacity];
        topIndex = -1;
    }

    void push(int x) {
        if (topIndex == capacity - 1) return;
        arr[++topIndex] = x;
    }

    int pop() {
        if (topIndex == -1) return -1;
        return arr[topIndex--];
    }
```

```
Top: 30
Pop: 30
Is Empty: 0
Is Full: 0


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

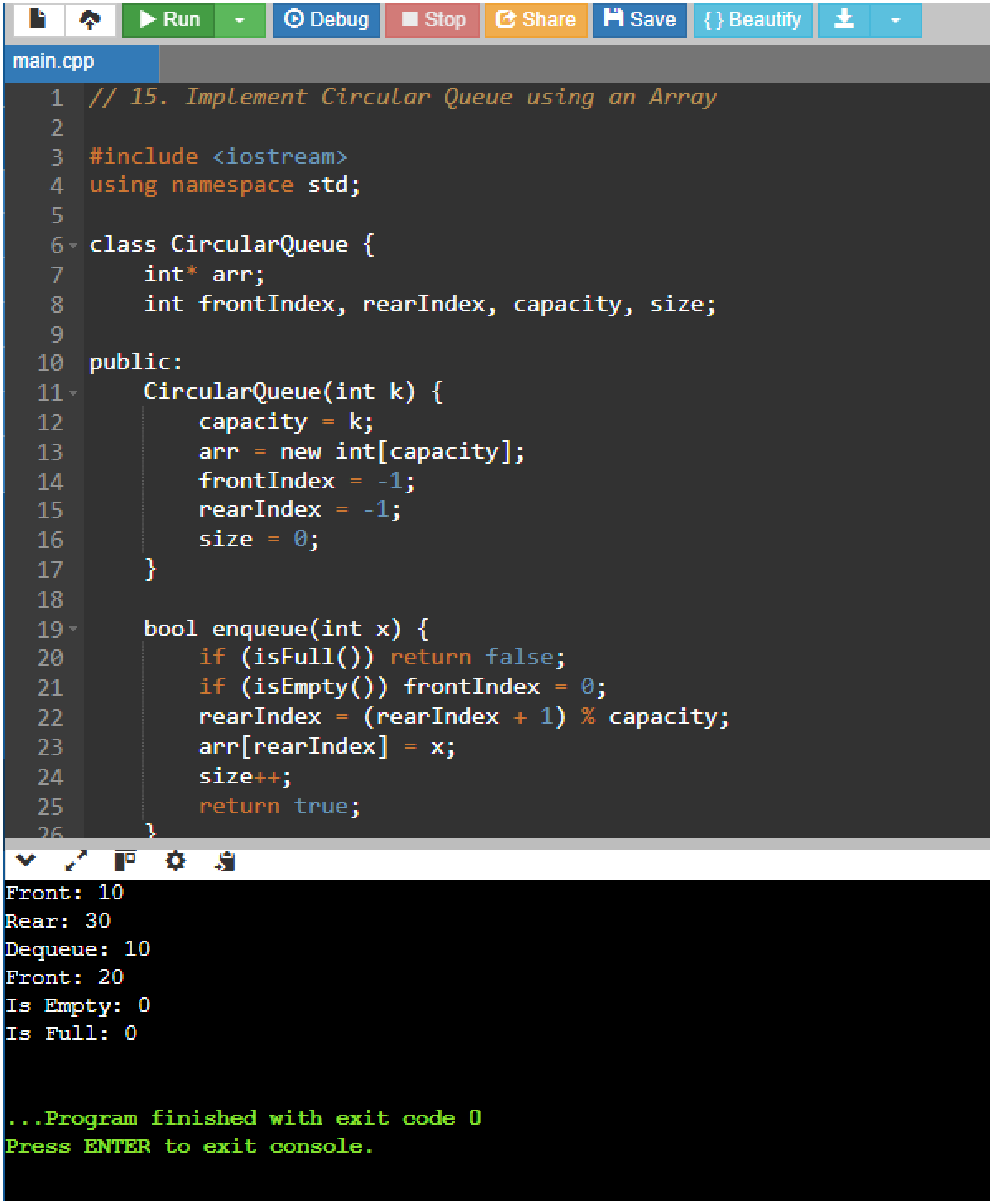
14. Implement Queue using an Array



```cpp
// 14. Implement Queue using an Array

#include <iostream>
using namespace std;

class Queue {
    int* arr;
    int frontIndex, rearIndex, capacity;

public:
    Queue(int size) {
        capacity = size;
        arr = new int[capacity];
        frontIndex = 0;
        rearIndex = -1;
    }

    void enqueue(int x) {
        if (rearIndex == capacity - 1) return;
        arr[++rearIndex] = x;
    }

    int dequeue() {
        if (frontIndex > rearIndex) return -1;
        return arr[frontIndex++];
    }
```

```
Front: 10
Dequeue: 10
Is Empty: 0
Is Full: 0


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

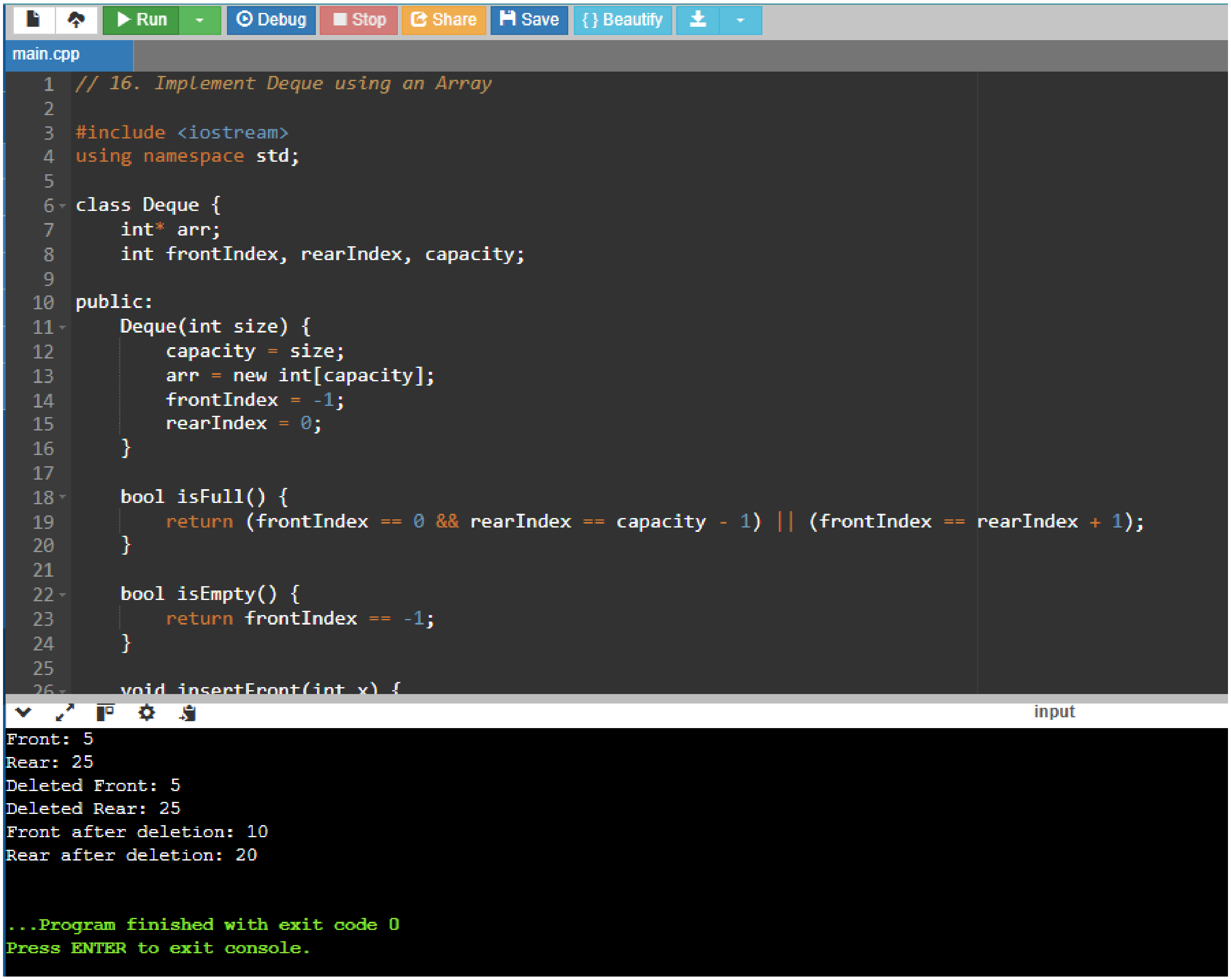
15. Implement Circular Queue using an Array



```cpp
// 15. Implement Circular Queue using an Array

#include <iostream>
using namespace std;

class CircularQueue {
    int* arr;
    int frontIndex, rearIndex, capacity, size;

public:
    CircularQueue(int k) {
        capacity = k;
        arr = new int[capacity];
        frontIndex = -1;
        rearIndex = -1;
        size = 0;
    }

    bool enqueue(int x) {
        if (isFull()) return false;
        if (isEmpty()) frontIndex = 0;
        rearIndex = (rearIndex + 1) % capacity;
        arr[rearIndex] = x;
        size++;
        return true;
    }
```

```
Front: 10
Rear: 30
Dequeue: 10
Front: 20
Is Empty: 0
Is Full: 0


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

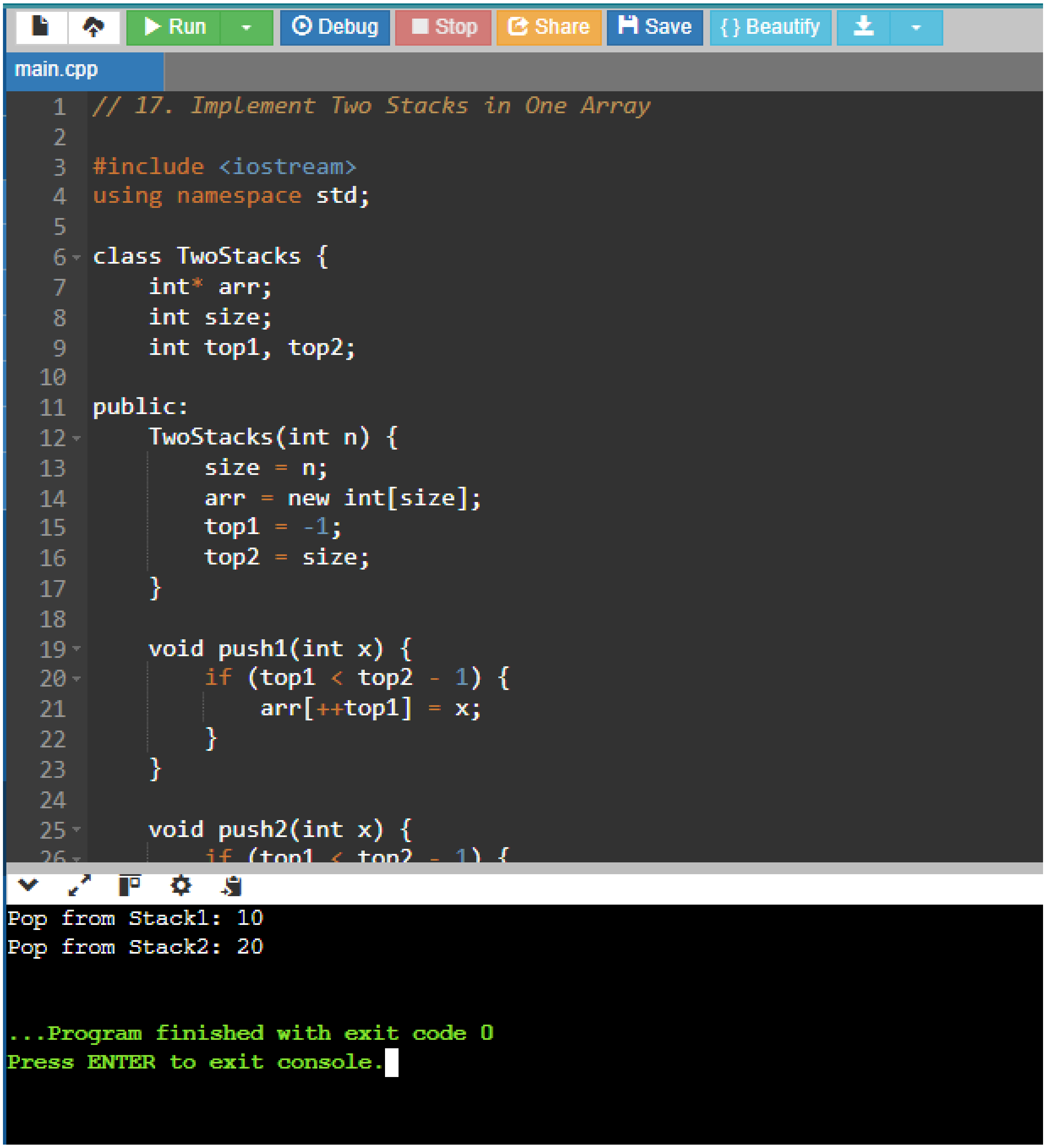
16. Implement Deque using an Array



```
// 16. Implement Deque using an Array

#include <iostream>
using namespace std;

class Deque {
    int* arr;
    int frontIndex, rearIndex, capacity;

public:
    Deque(int size) {
        capacity = size;
        arr = new int[capacity];
        frontIndex = -1;
        rearIndex = 0;
    }

    bool isFull() {
        return (frontIndex == 0 && rearIndex == capacity - 1) || (frontIndex == rearIndex + 1);
    }

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

    void insertFront(int x) {
```

```
Front: 5
Rear: 25
Deleted Front: 5
Deleted Rear: 25
Front after deletion: 10
Rear after deletion: 20


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

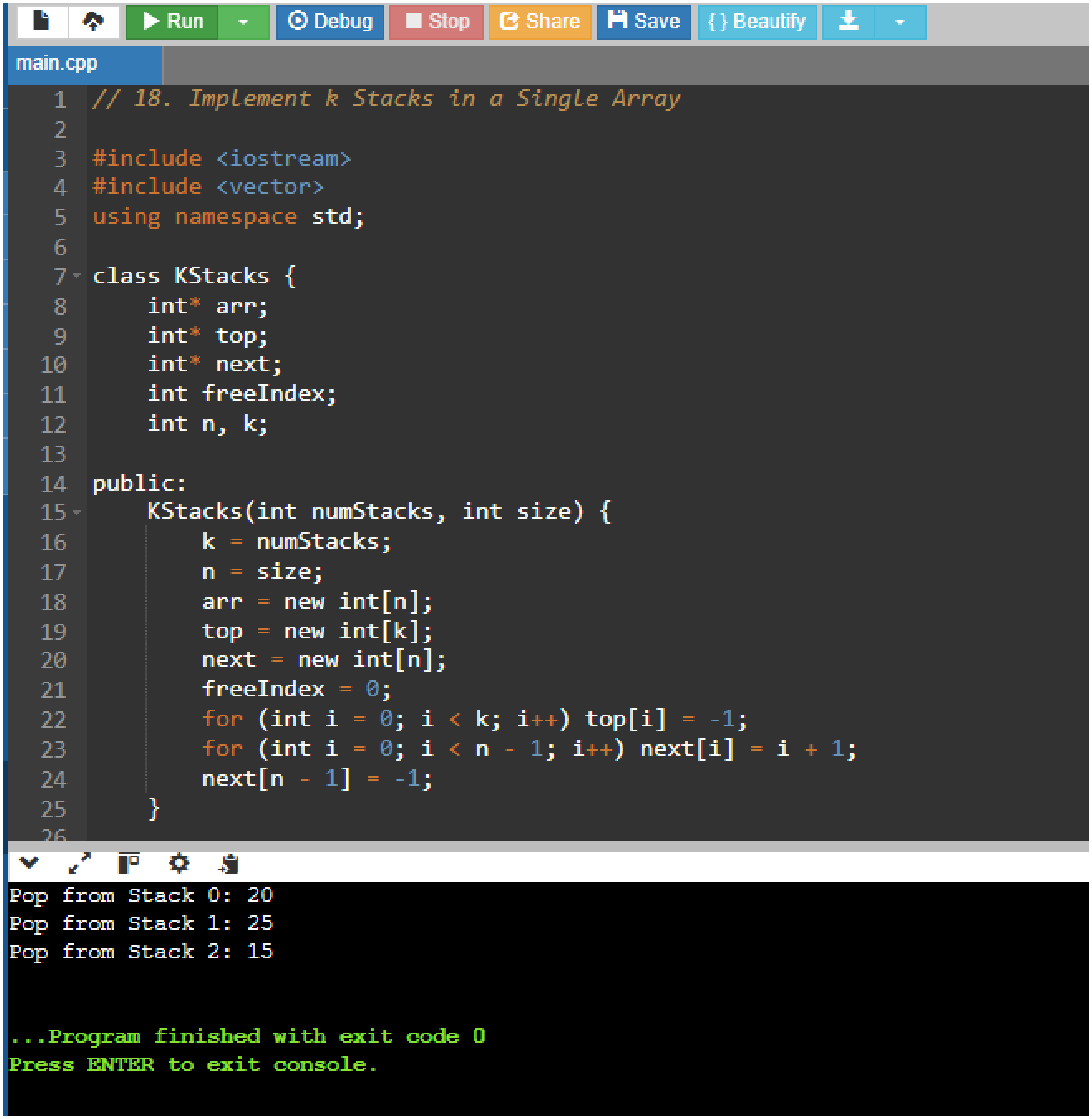
17. Implement Two Stacks in One Array



```cpp
// 17. Implement Two Stacks in One Array

#include <iostream>
using namespace std;

class TwoStacks {
    int* arr;
    int size;
    int top1, top2;

public:
    TwoStacks(int n) {
        size = n;
        arr = new int[size];
        top1 = -1;
        top2 = size;
    }

    void push1(int x) {
        if (top1 < top2 - 1) {
            arr[++top1] = x;
        }
    }

    void push2(int x) {
        if (top1 < top2 - 1) {
```

```
Pop from Stack1: 10
Pop from Stack2: 20


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

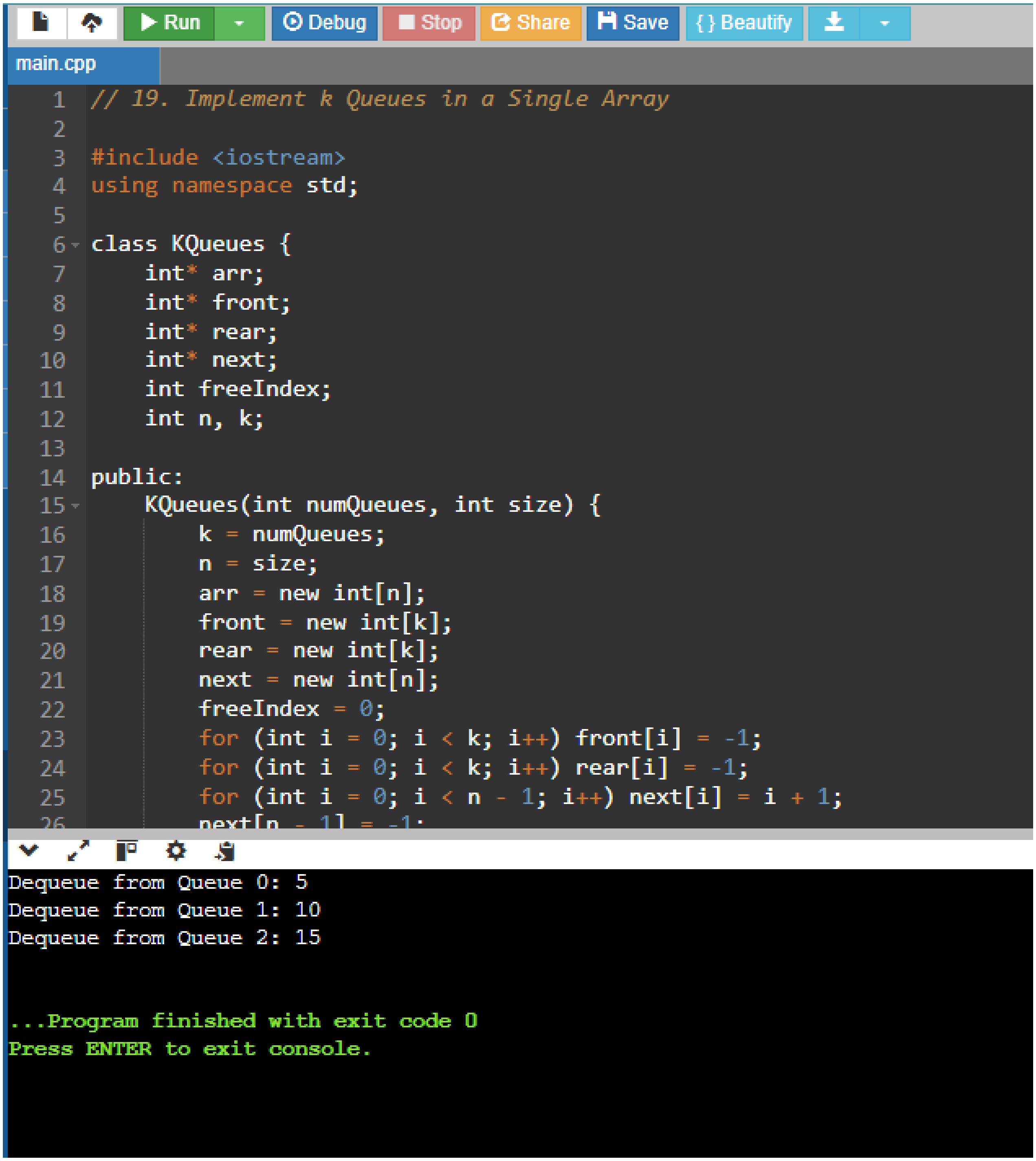
18. Implement k Stacks in a Single Array



```cpp
// 18. Implement k Stacks in a Single Array

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

class KStacks {
    int* arr;
    int* top;
    int* next;
    int freeIndex;
    int n, k;

public:
    KStacks(int numStacks, int size) {
        k = numStacks;
        n = size;
        arr = new int[n];
        top = new int[k];
        next = new int[n];
        freeIndex = 0;
        for (int i = 0; i < k; i++) top[i] = -1;
        for (int i = 0; i < n - 1; i++) next[i] = i + 1;
        next[n - 1] = -1;
    }
```

```
Pop from Stack 0: 20
Pop from Stack 1: 25
Pop from Stack 2: 15


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

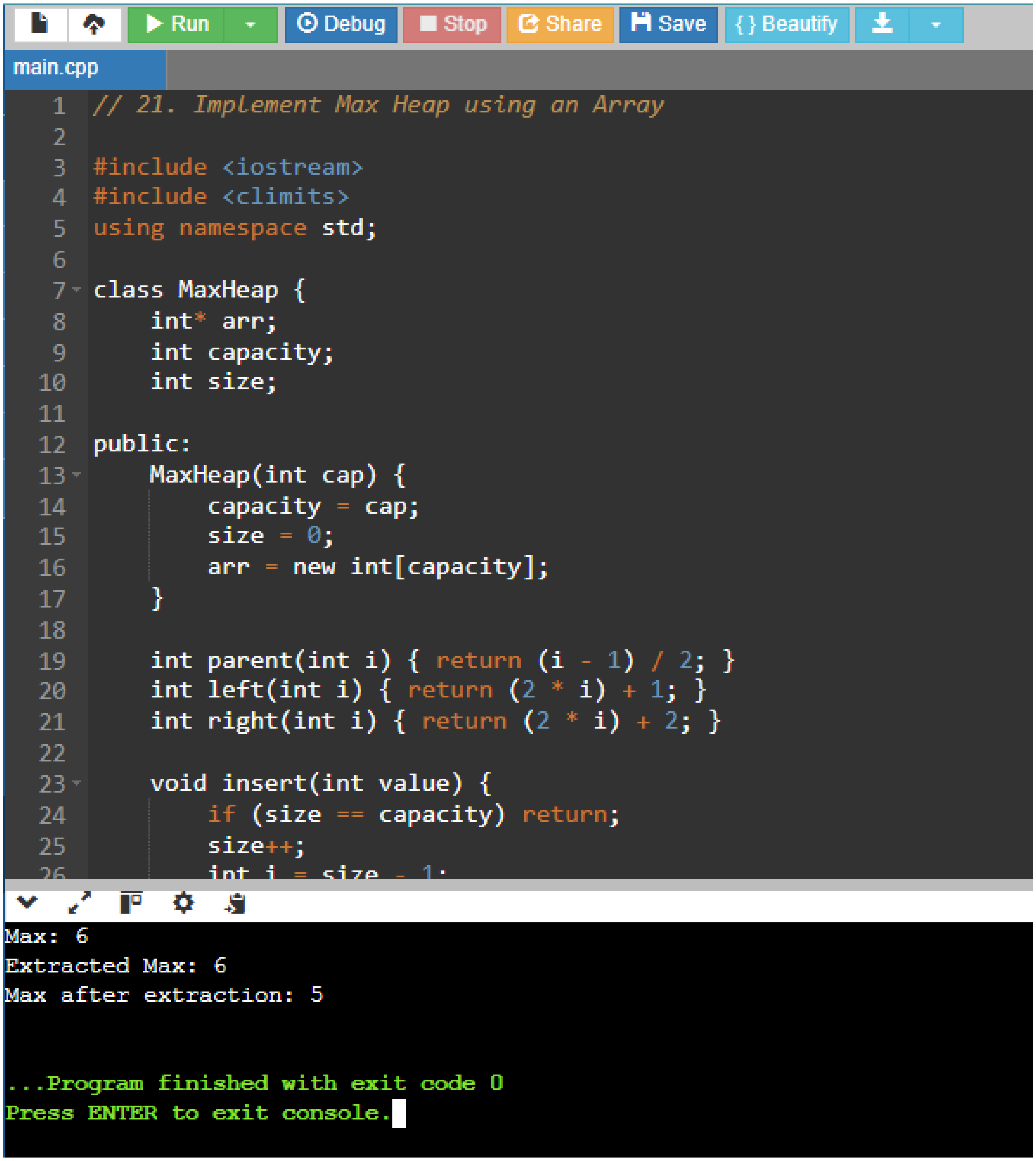
19. Implement k Queues in a Single Array



```cpp
// 19. Implement k Queues in a Single Array

#include <iostream>
using namespace std;

class KQueues {
    int* arr;
    int* front;
    int* rear;
    int* next;
    int freeIndex;
    int n, k;

public:
    KQueues(int numQueues, int size) {
        k = numQueues;
        n = size;
        arr = new int[n];
        front = new int[k];
        rear = new int[k];
        next = new int[n];
        freeIndex = 0;
        for (int i = 0; i < k; i++) front[i] = -1;
        for (int i = 0; i < k; i++) rear[i] = -1;
        for (int i = 0; i < n - 1; i++) next[i] = i + 1;
        next[n - 1] = -1;
```

```
Dequeue from Queue 0: 5
Dequeue from Queue 1: 10
Dequeue from Queue 2: 15


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

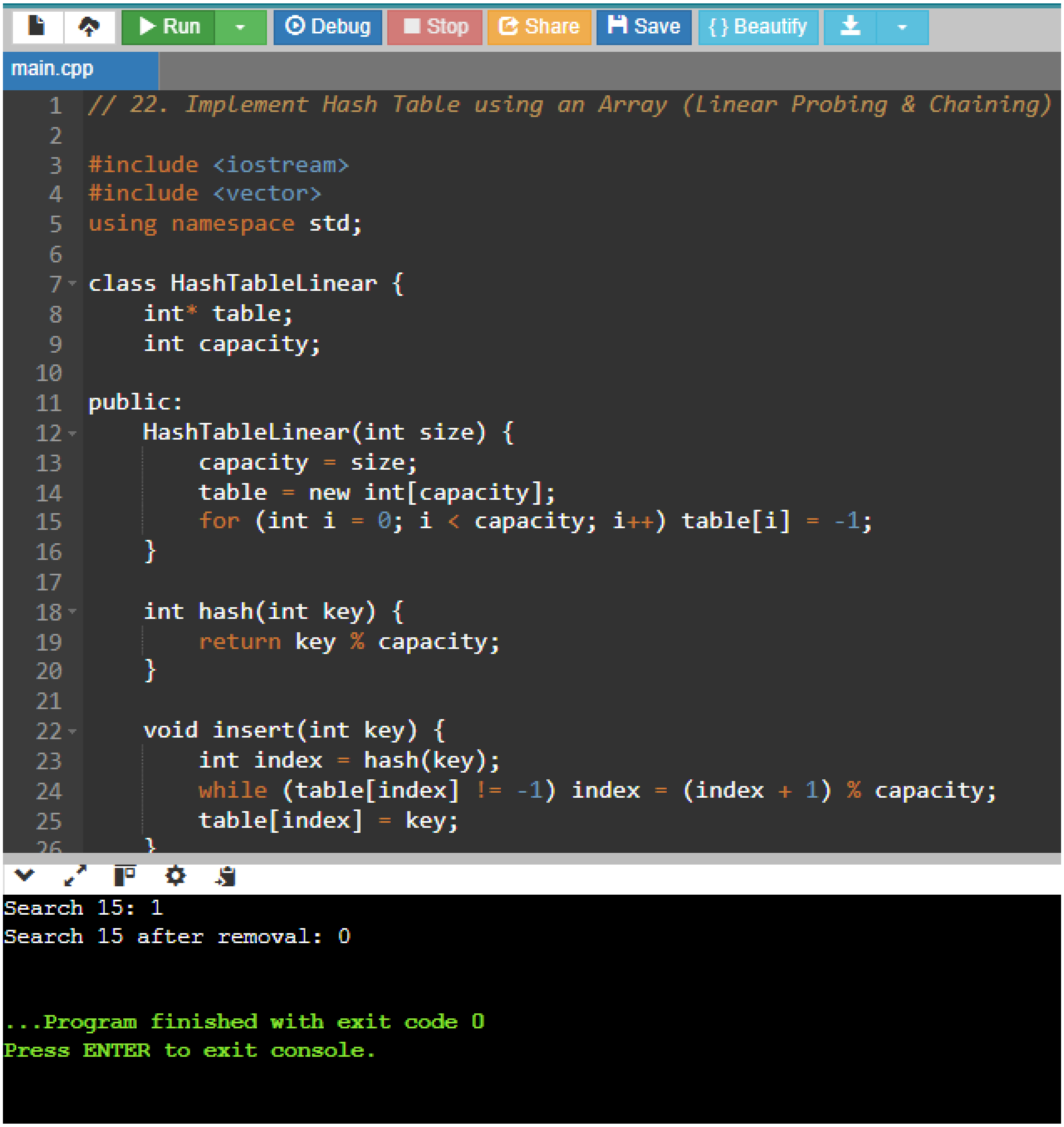
21. Implement Max Heap using an Array



```cpp
// 21. Implement Max Heap using an Array

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

class MaxHeap {
    int* arr;
    int capacity;
    int size;

public:
    MaxHeap(int cap) {
        capacity = cap;
        size = 0;
        arr = new int[capacity];
    }

    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return (2 * i) + 1; }
    int right(int i) { return (2 * i) + 2; }

    void insert(int value) {
        if (size == capacity) return;
        size++;
        int i = size - 1;
```

```
Max: 6
Extracted Max: 6
Max after extraction: 5


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

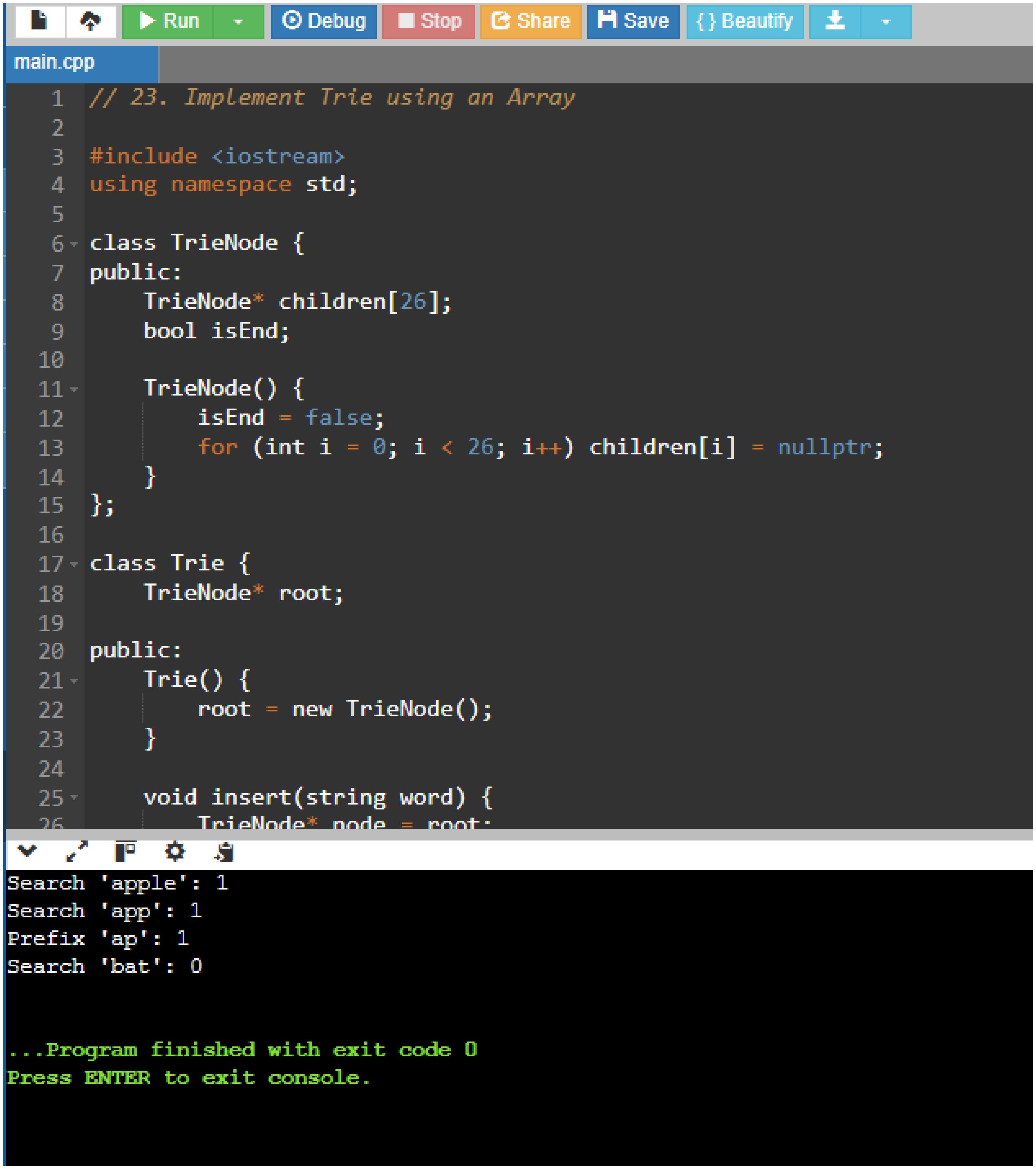
22. Implement Hash Table using an Array (Linear Probing & Chaining)



```cpp
// 22. Implement Hash Table using an Array (Linear Probing & Chaining)

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

class HashTableLinear {
    int* table;
    int capacity;

public:
    HashTableLinear(int size) {
        capacity = size;
        table = new int[capacity];
        for (int i = 0; i < capacity; i++) table[i] = -1;
    }

    int hash(int key) {
        return key % capacity;
    }

    void insert(int key) {
        int index = hash(key);
        while (table[index] != -1) index = (index + 1) % capacity;
        table[index] = key;
    }
```

```
Search 15: 1
Search 15 after removal: 0


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

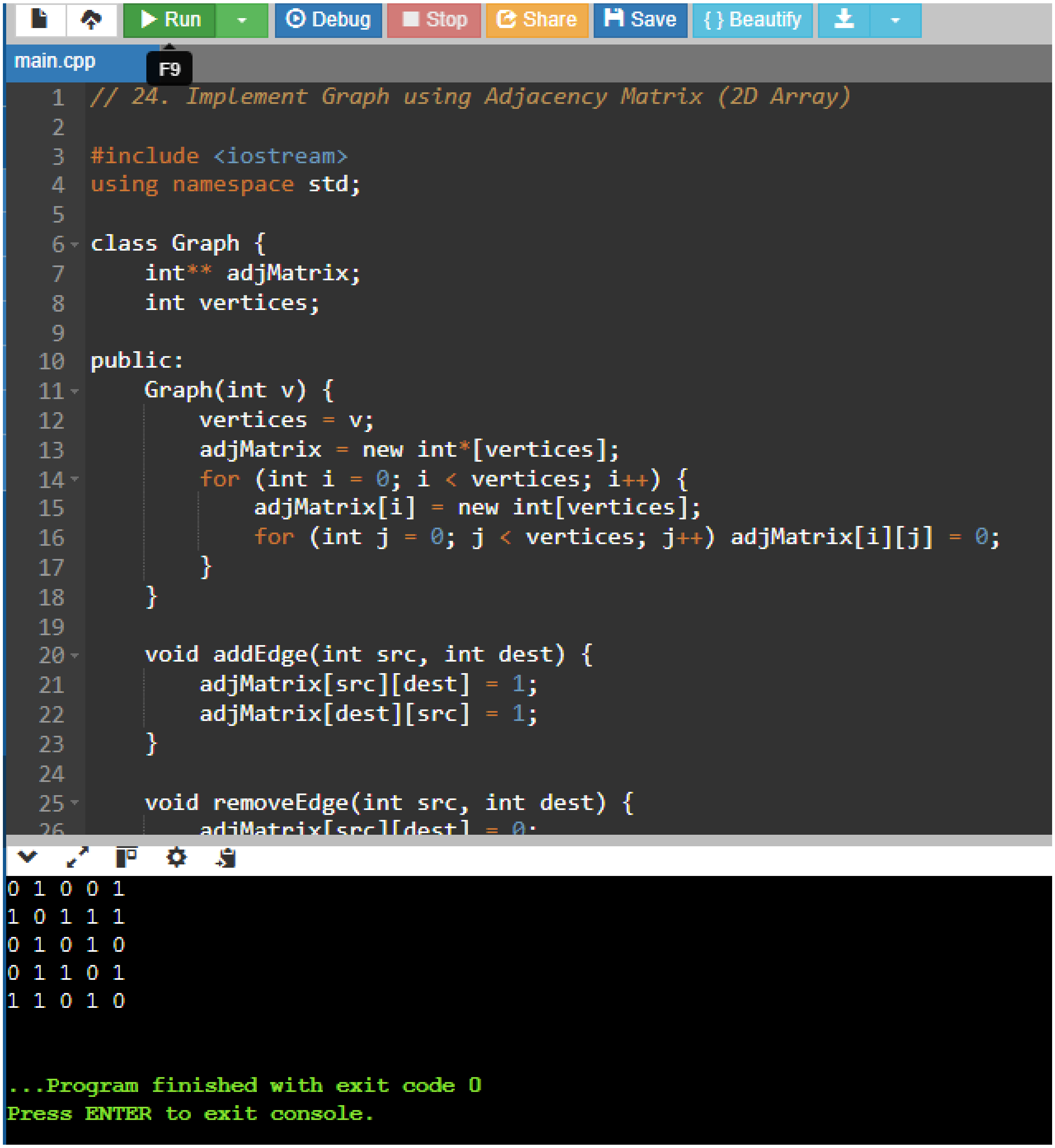
23. Implement Trie using an Array



```cpp
// 23. Implement Trie using an Array

#include <iostream>
using namespace std;

class TrieNode {
public:
    TrieNode* children[26];
    bool isEnd;

    TrieNode() {
        isEnd = false;
        for (int i = 0; i < 26; i++) children[i] = nullptr;
    }
};

class Trie {
    TrieNode* root;

public:
    Trie() {
        root = new TrieNode();
    }

    void insert(string word) {
        TrieNode* node = root;
```

```
Search 'apple': 1
Search 'app': 1
Prefix 'ap': 1
Search 'bat': 0


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

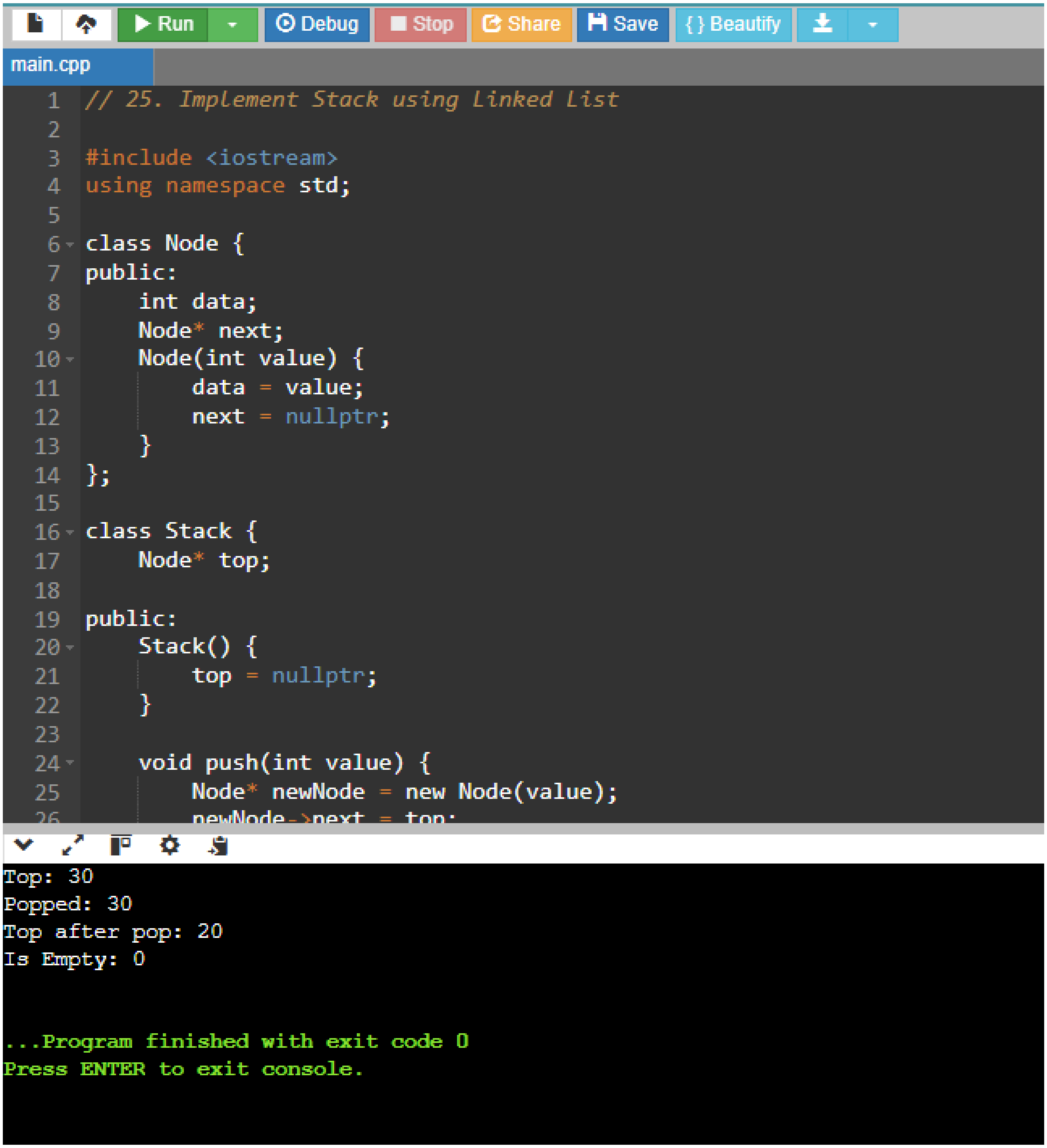
24. Implement Graph using Adjacency Matrix (2D Array)



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

#include <iostream>
using namespace std;

class Graph {
    int** adjMatrix;
    int vertices;

public:
    Graph(int v) {
        vertices = v;
        adjMatrix = new int*[vertices];
        for (int i = 0; i < vertices; i++) {
            adjMatrix[i] = new int[vertices];
            for (int j = 0; j < vertices; j++) adjMatrix[i][j] = 0;
        }
    }

    void addEdge(int src, int dest) {
        adjMatrix[src][dest] = 1;
        adjMatrix[dest][src] = 1;
    }

    void removeEdge(int src, int dest) {
        adjMatrix[src][dest] = 0;
```

```
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


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

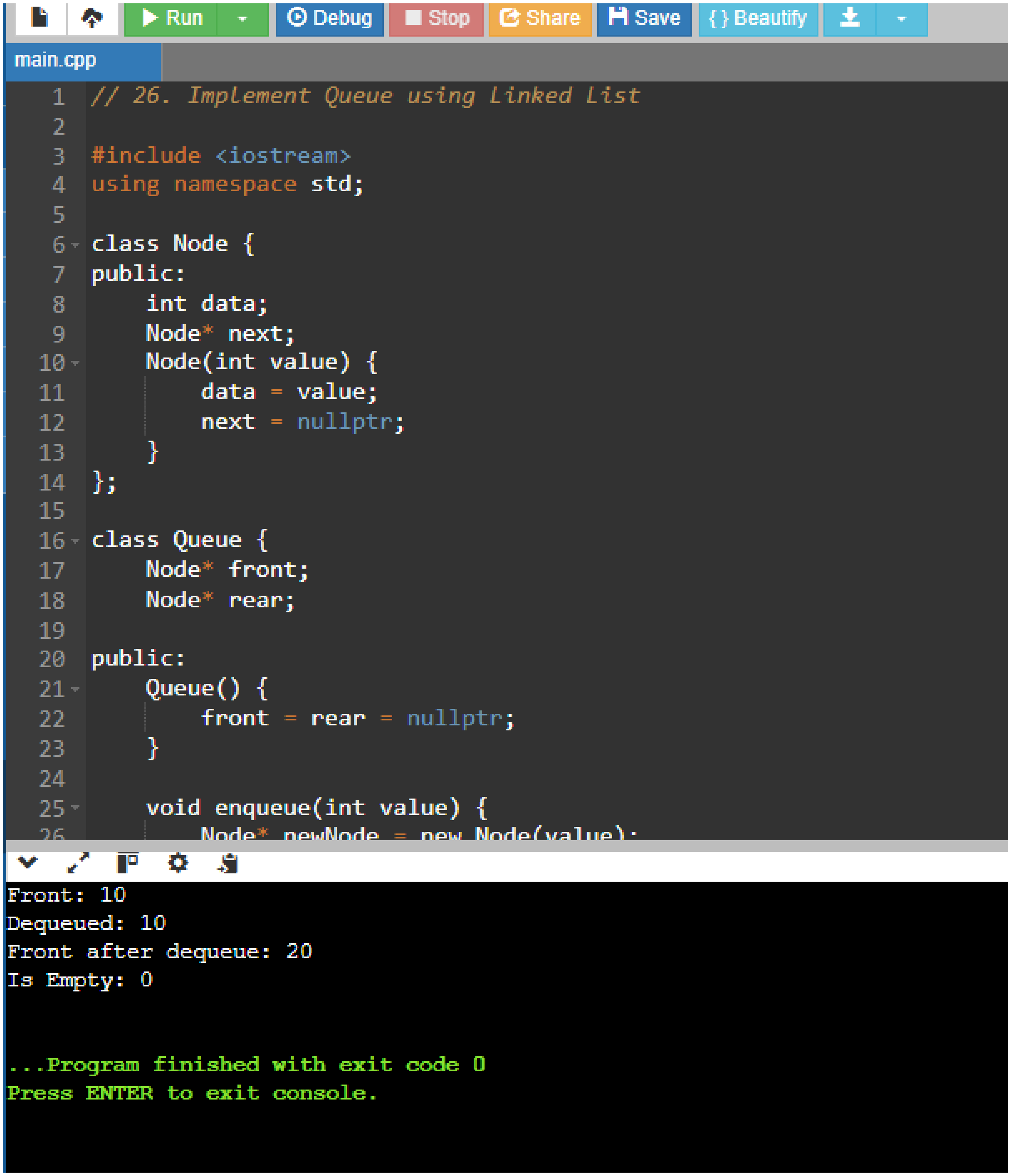
Q 25. Implement Stack using Linked List



```cpp
// 25. Implement Stack using Linked List

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};

class Stack {
    Node* top;

public:
    Stack() {
        top = nullptr;
    }

    void push(int value) {
        Node* newNode = new Node(value);
        newNode->next = top;
```

```
Top: 30
Popped: 30
Top after pop: 20
Is Empty: 0


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

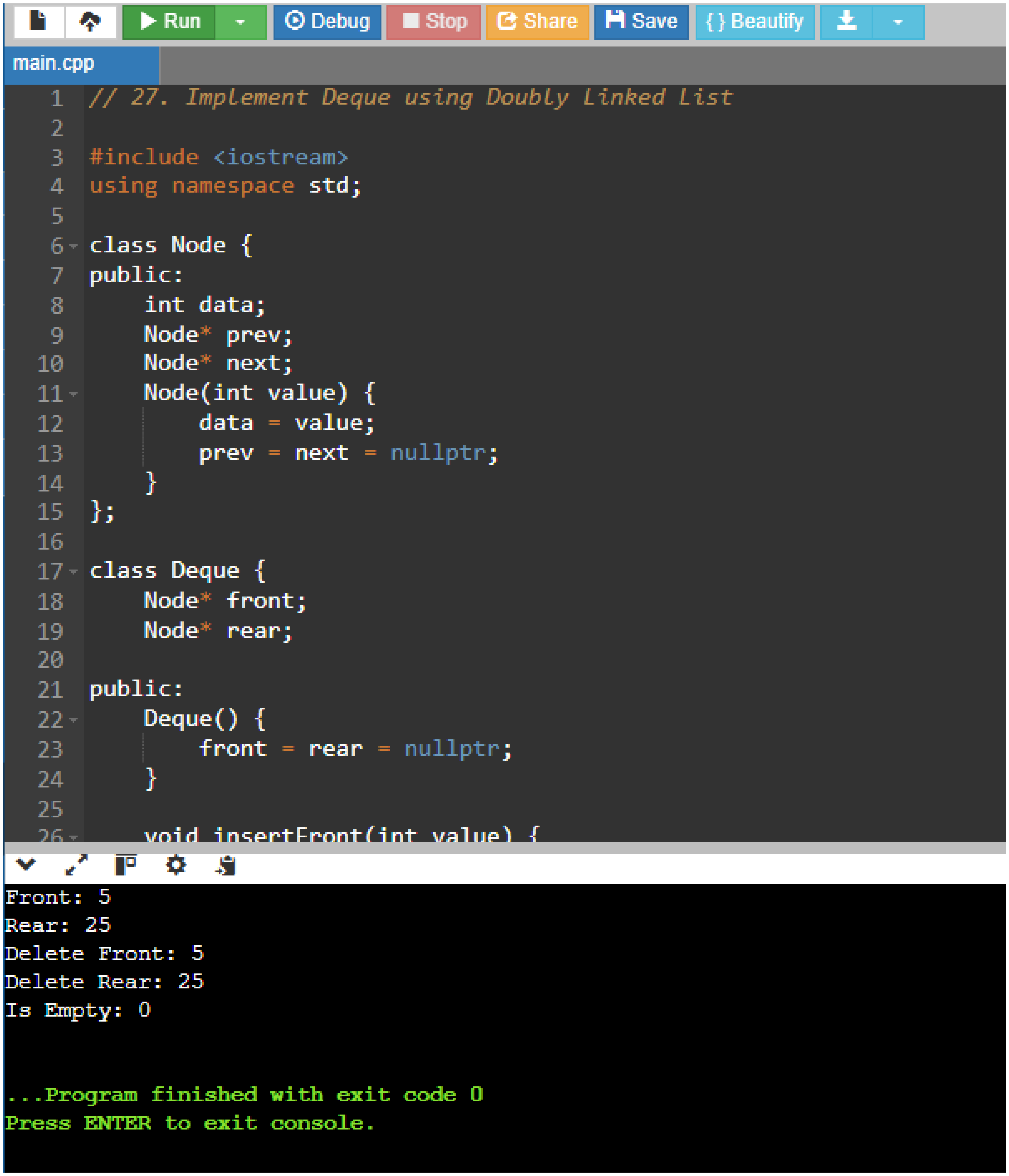
26. Implement Queue using Linked List



```cpp
// 26. Implement Queue using Linked List

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};

class Queue {
    Node* front;
    Node* rear;

public:
    Queue() {
        front = rear = nullptr;
    }

    void enqueue(int value) {
        Node* newNode = new Node(value);
```

```
Front: 10
Dequeued: 10
Front after dequeue: 20
Is Empty: 0


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

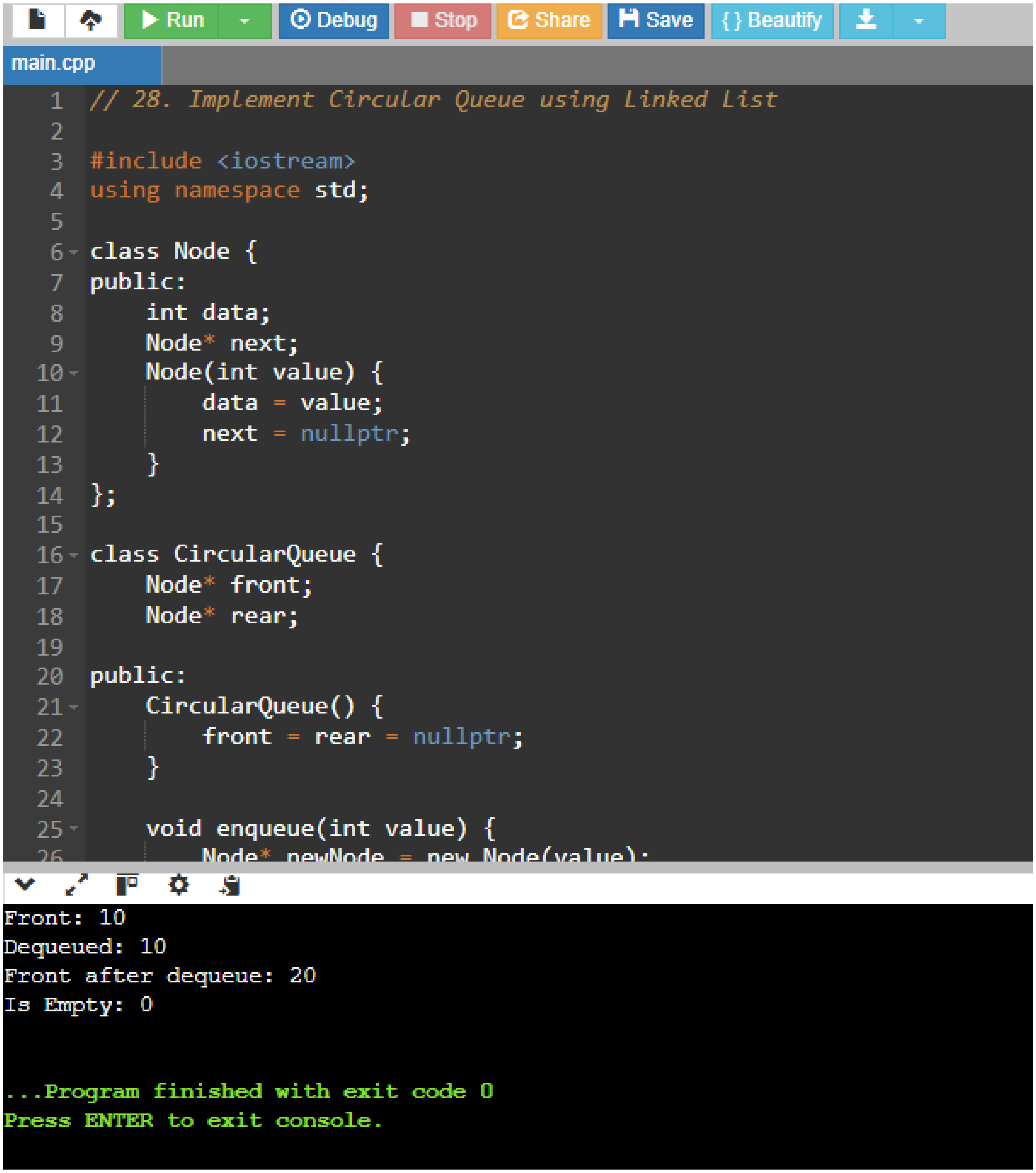
27. Implement Deque using Doubly Linked List



```cpp
// 27. Implement Deque using Doubly Linked List

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;
    Node(int value) {
        data = value;
        prev = next = nullptr;
    }
};

class Deque {
    Node* front;
    Node* rear;

public:
    Deque() {
        front = rear = nullptr;
    }

    void insertFront(int value) {
```

```
Front: 5
Rear: 25
Delete Front: 5
Delete Rear: 25
Is Empty: 0


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

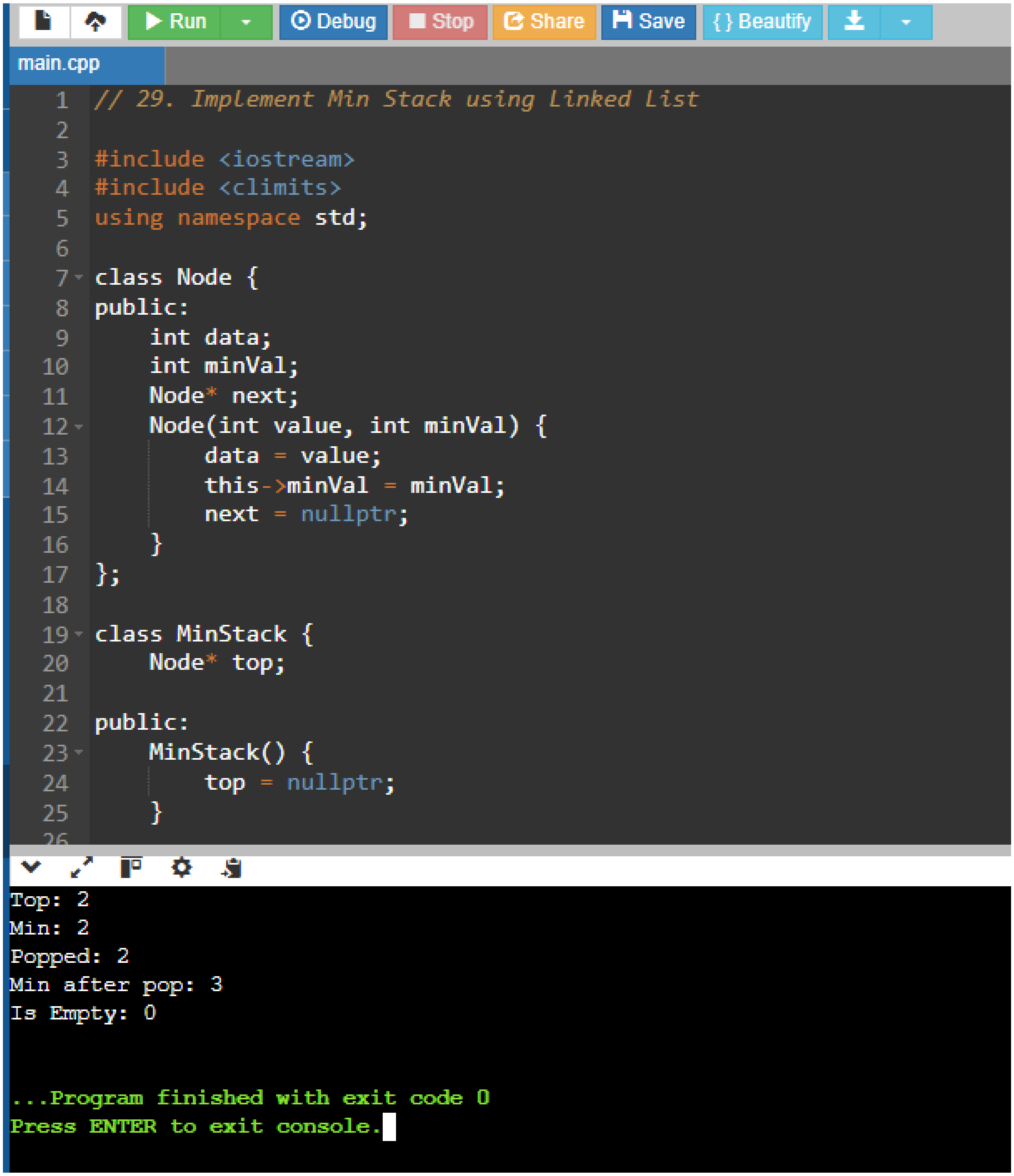
28. Implement Circular Queue using Linked List



```cpp
// 28. Implement Circular Queue using Linked List

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};

class CircularQueue {
    Node* front;
    Node* rear;

public:
    CircularQueue() {
        front = rear = nullptr;
    }

    void enqueue(int value) {
        Node* newNode = new Node(value);
```

```
Front: 10
Dequeued: 10
Front after dequeue: 20
Is Empty: 0


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

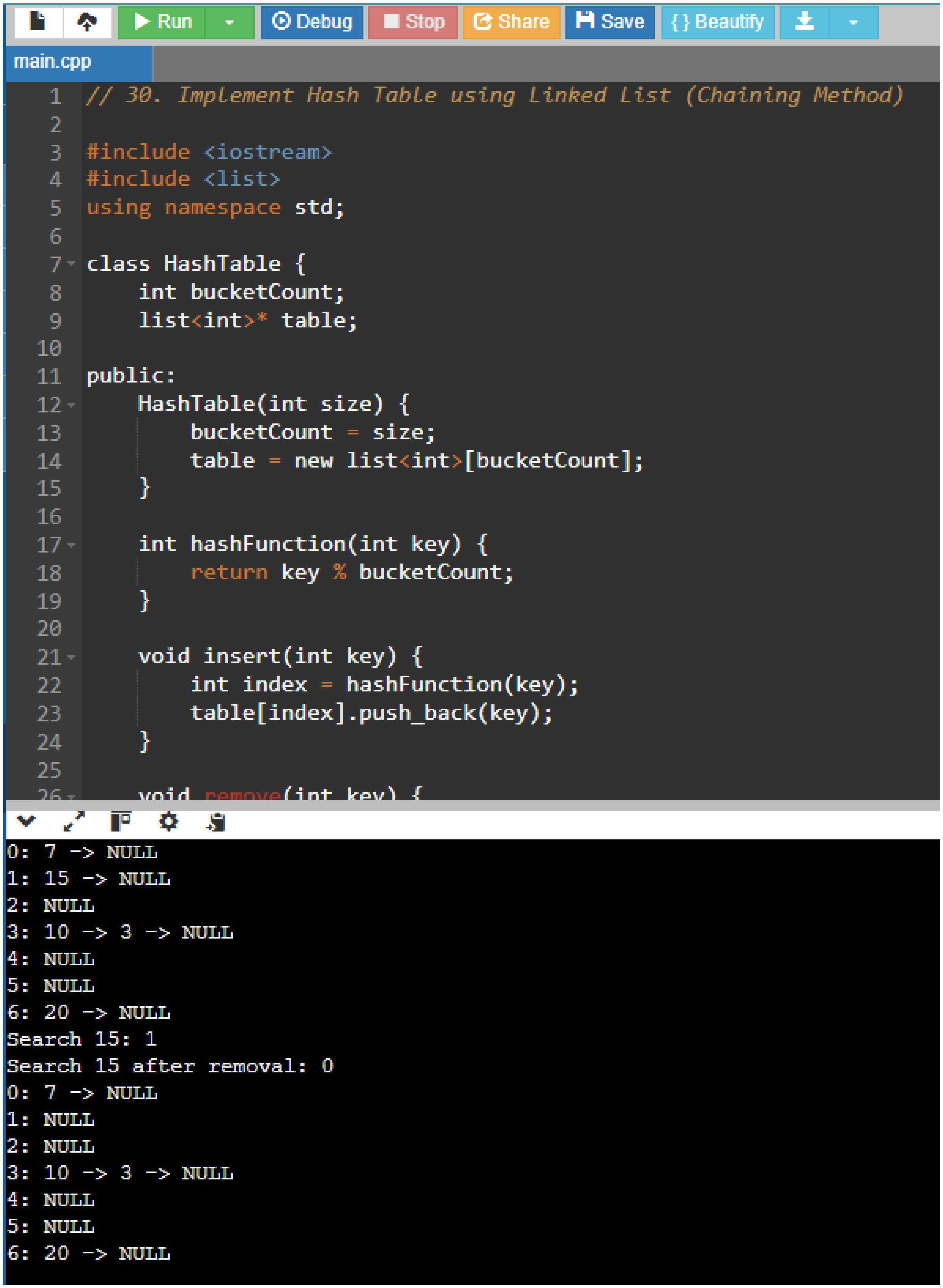
29. Implement Min Stack using Linked List



```cpp
// 29. Implement Min Stack using Linked List

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

class Node {
public:
    int data;
    int minVal;
    Node* next;
    Node(int value, int minVal) {
        data = value;
        this->minVal = minVal;
        next = nullptr;
    }
};

class MinStack {
    Node* top;

public:
    MinStack() {
        top = nullptr;
    }
```

```
Top: 2
Min: 2
Popped: 2
Min after pop: 3
Is Empty: 0


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

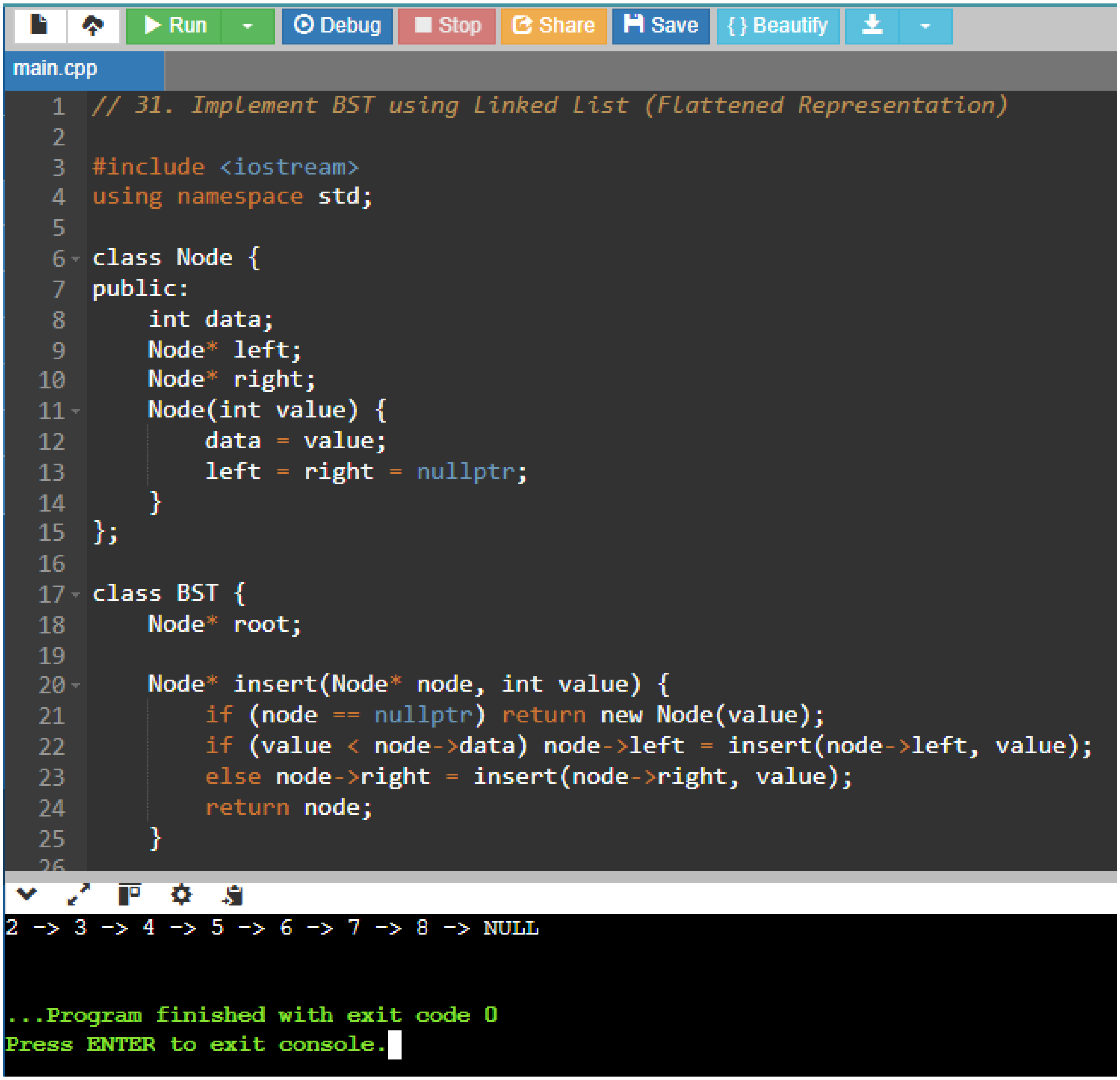
30. Implement Hash Table using Linked List (Chaining Method)



```cpp
// 30. Implement Hash Table using Linked List (Chaining Method)

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

class HashTable {
    int bucketCount;
    list<int>* table;

public:
    HashTable(int size) {
        bucketCount = size;
        table = new list<int>[bucketCount];
    }

    int hashFunction(int key) {
        return key % bucketCount;
    }

    void insert(int key) {
        int index = hashFunction(key);
        table[index].push_back(key);
    }

    void remove(int key) {
```

```
0: 7 -> NULL
1: 15 -> NULL
2: NULL
3: 10 -> 3 -> NULL
4: NULL
5: NULL
6: 20 -> NULL
Search 15: 1
Search 15 after removal: 0
0: 7 -> NULL
1: NULL
2: NULL
3: 10 -> 3 -> NULL
4: NULL
5: NULL
6: 20 -> NULL
```

31. Implement BST using Linked List (Flattened Representation)



```cpp
// 31. Implement BST using Linked List (Flattened Representation)

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

class BST {
    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;
    }
```

```
2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL


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