

Experiment 6

Student Name: Saurav Ashiwal

Branch: CSE

Semester: 6

Subject Name: AP lab-2

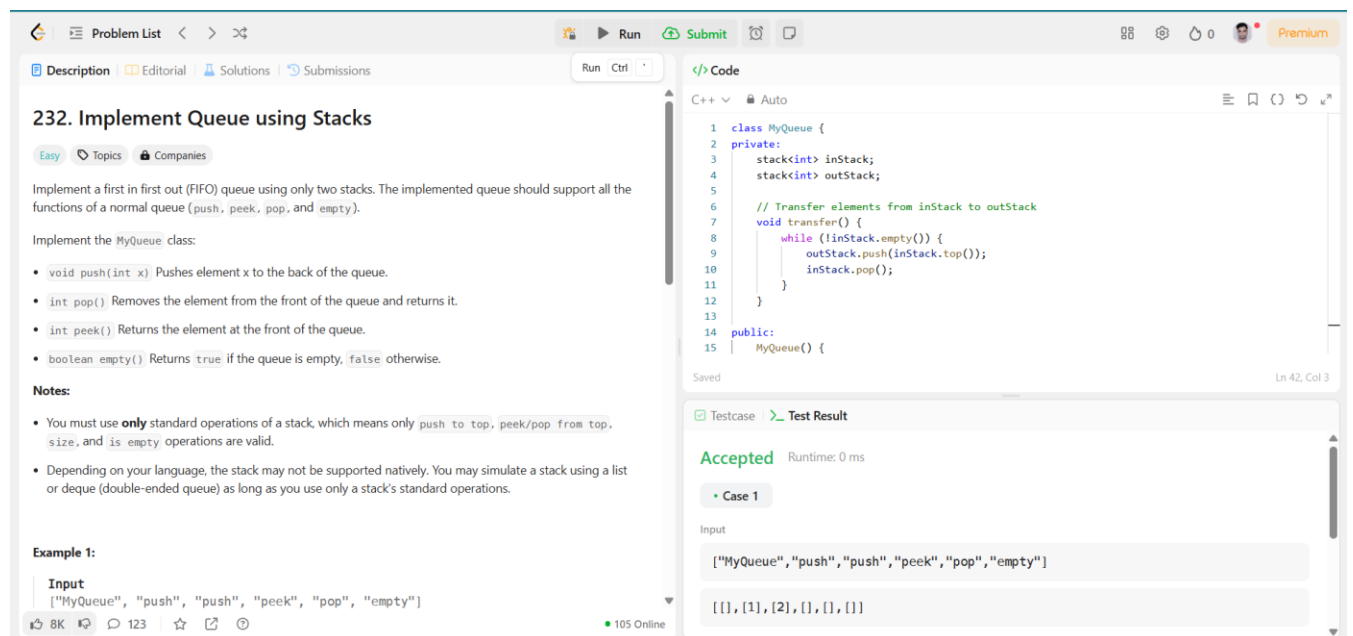
UID: 22BCS13250

Section/Group: 614/B

Date of Performance: 12/03/2025

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"]
```

Output

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

```
1 class MyQueue {
2     private:
3         stack<int> inStack;
4         stack<int> outStack;
5
6         // Transfer elements from inStack to outStack
7         void transfer() {
8             while (!inStack.empty()) {
9                 outStack.push(inStack.top());
10                inStack.pop();
11            }
12        }
13
14     public:
15         MyQueue() {}
```

Accepted Runtime: 0 ms

Case 1

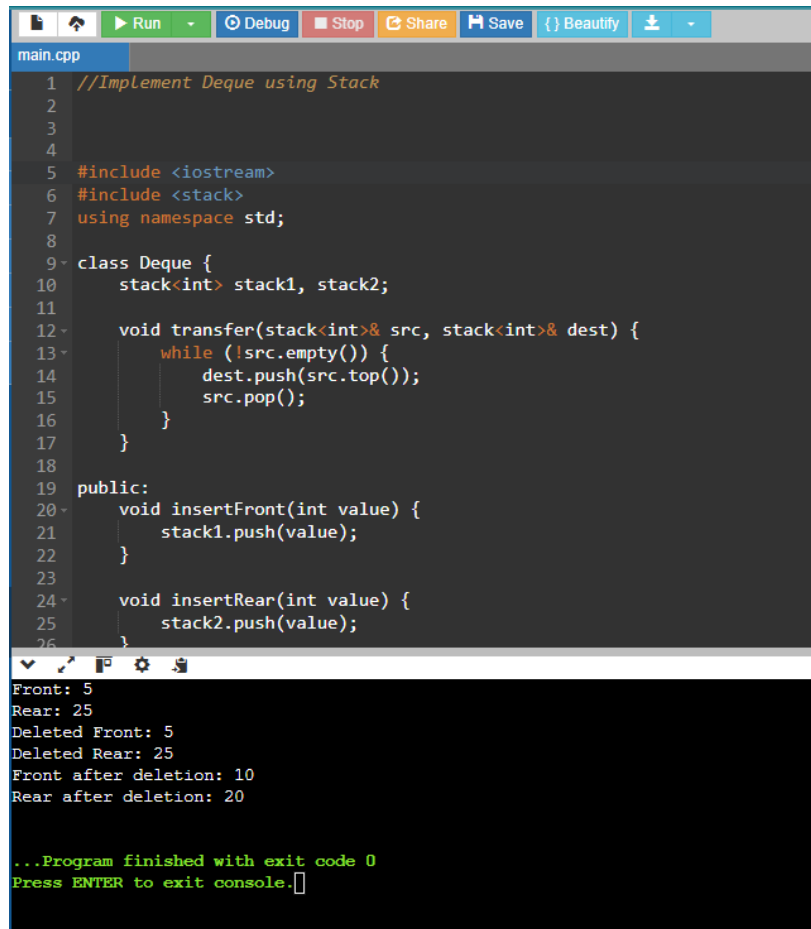
Input

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

Output

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

2.Implement Deque using Stack

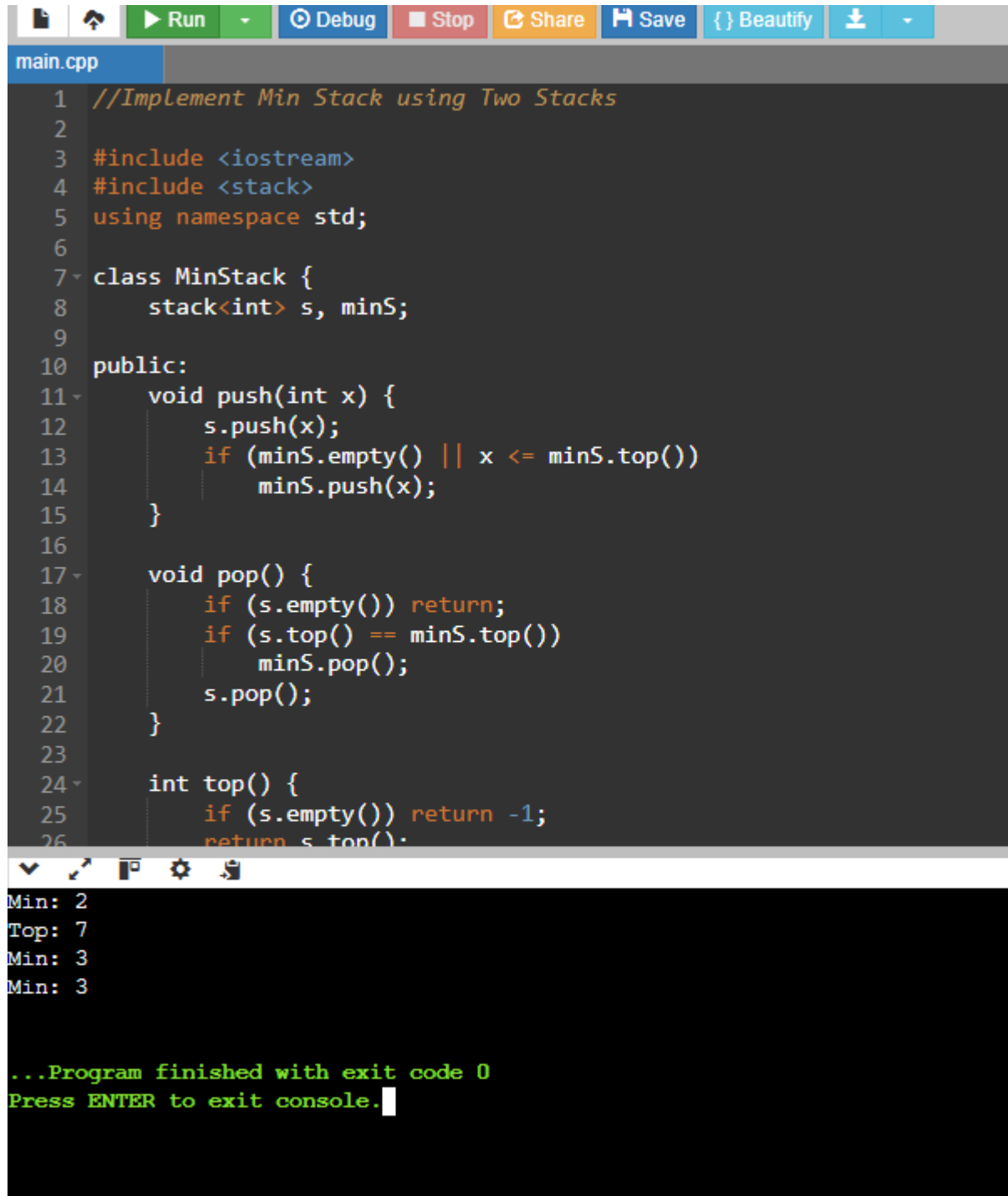


```
1 //Implement Deque using Stack
2
3
4
5 #include <iostream>
6 #include <stack>
7 using namespace std;
8
9 class Deque {
10     stack<int> stack1, stack2;
11
12     void transfer(stack<int>& src, stack<int>& dest) {
13         while (!src.empty()) {
14             dest.push(src.top());
15             src.pop();
16         }
17     }
18
19 public:
20     void insertFront(int value) {
21         stack1.push(value);
22     }
23
24     void insertRear(int value) {
25         stack2.push(value);
26     }
27 }
```

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

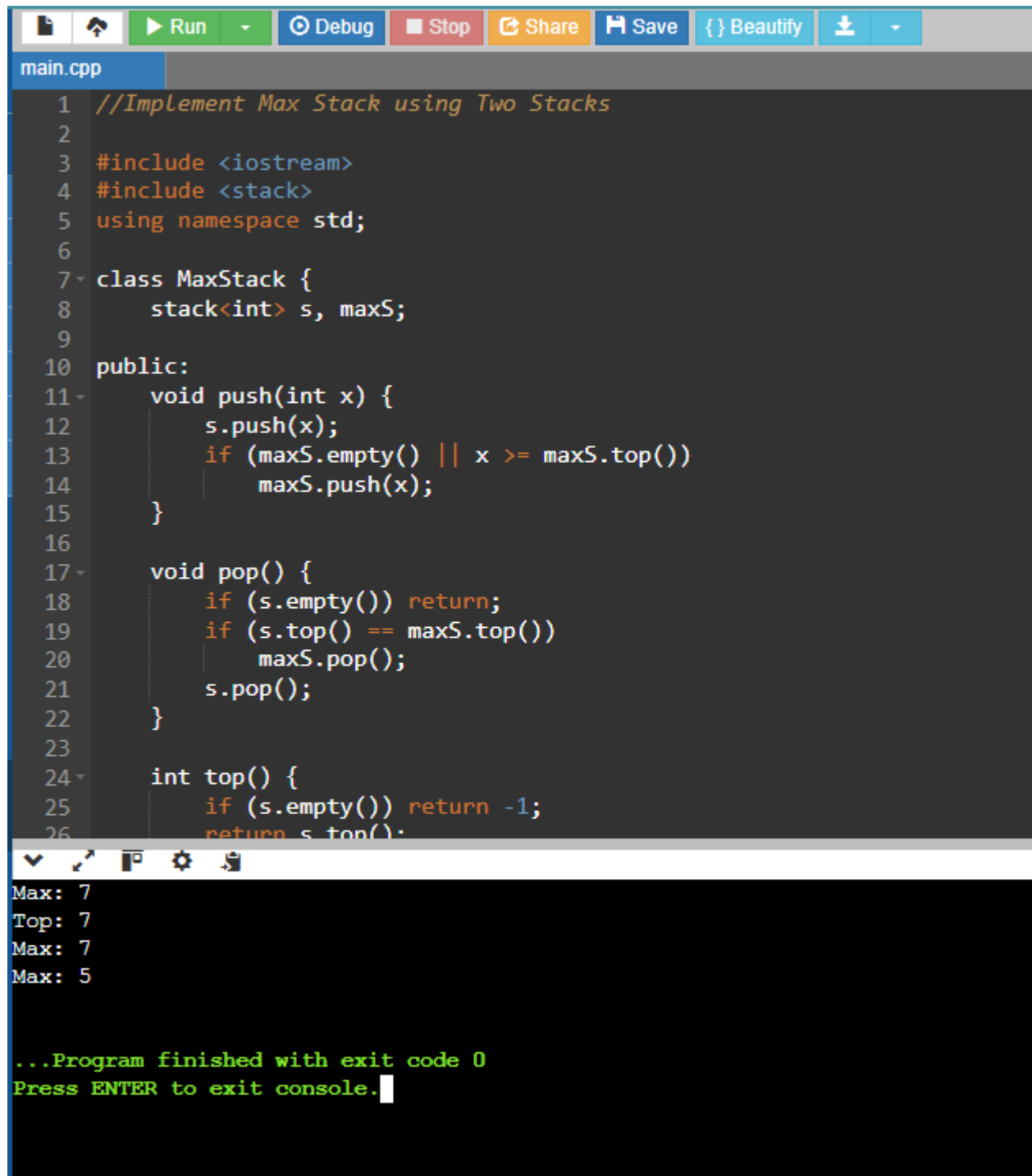


```
1 //Implement Min Stack using Two Stacks
2
3 #include <iostream>
4 #include <stack>
5 using namespace std;
6
7 class MinStack {
8     stack<int> s, minS;
9
10 public:
11     void push(int x) {
12         s.push(x);
13         if (minS.empty() || x <= minS.top())
14             minS.push(x);
15     }
16
17     void pop() {
18         if (s.empty()) return;
19         if (s.top() == minS.top())
20             minS.pop();
21         s.pop();
22     }
23
24     int top() {
25         if (s.empty()) return -1;
26         return s.top();
27     }
28 }
```

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

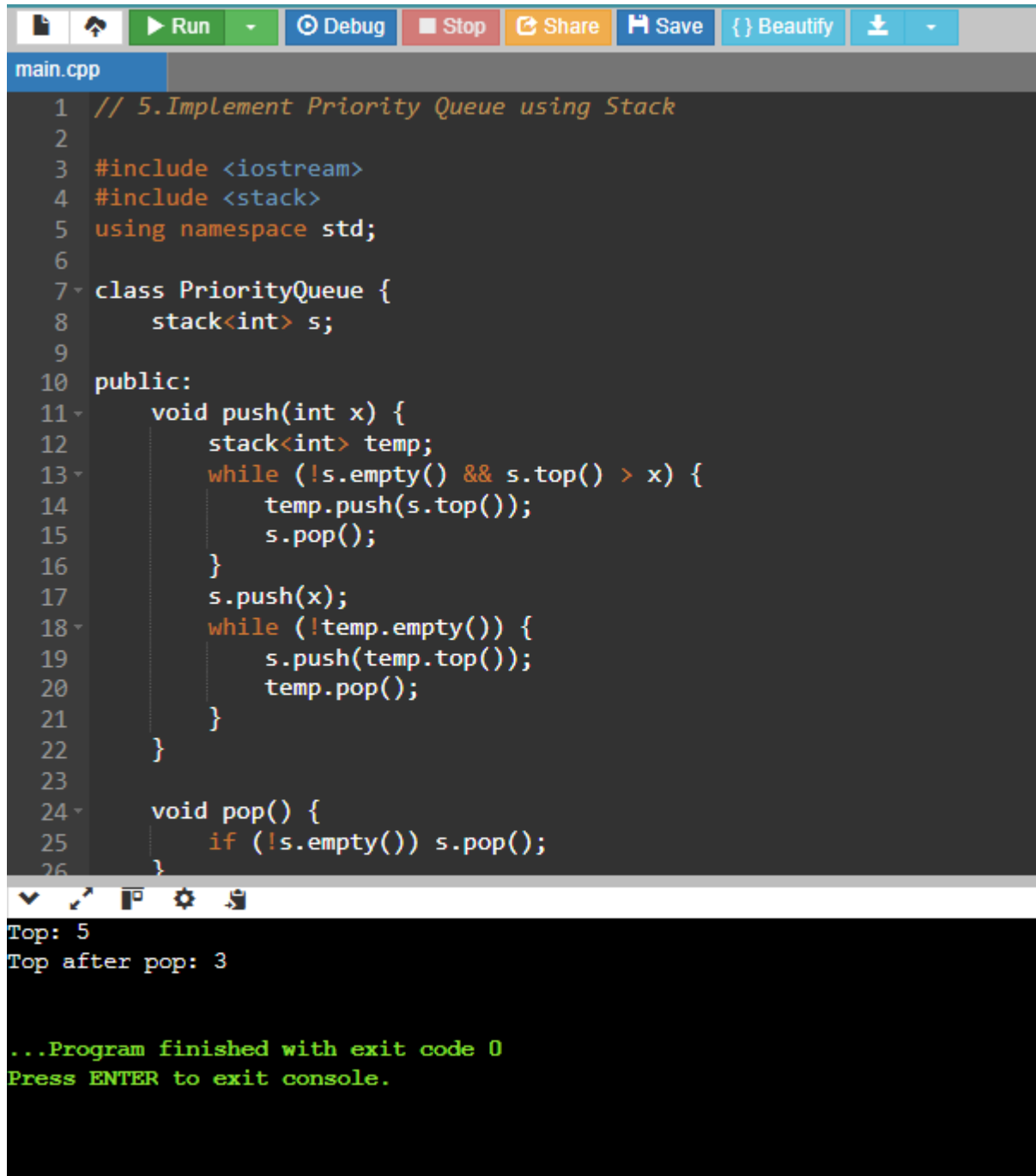


```
main.cpp
1 //Implement Max Stack using Two Stacks
2
3 #include <iostream>
4 #include <stack>
5 using namespace std;
6
7 class MaxStack {
8     stack<int> s, maxS;
9
10 public:
11     void push(int x) {
12         s.push(x);
13         if (maxS.empty() || x >= maxS.top())
14             maxS.push(x);
15     }
16
17     void pop() {
18         if (s.empty()) return;
19         if (s.top() == maxS.top())
20             maxS.pop();
21         s.pop();
22     }
23
24     int top() {
25         if (s.empty()) return -1;
26         return s.top();
27     }
28 }
29
30 int main() {
31     MaxStack ms;
32     ms.push(7);
33     ms.push(7);
34     ms.push(5);
35     ms.pop();
36     cout << "Max: " << ms.top() << endl;
37     cout << "Top: " << ms.top() << endl;
38     cout << "Max: " << ms.top() << endl;
39     cout << "Max: " << ms.top() << endl;
40     return 0;
41 }
```

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

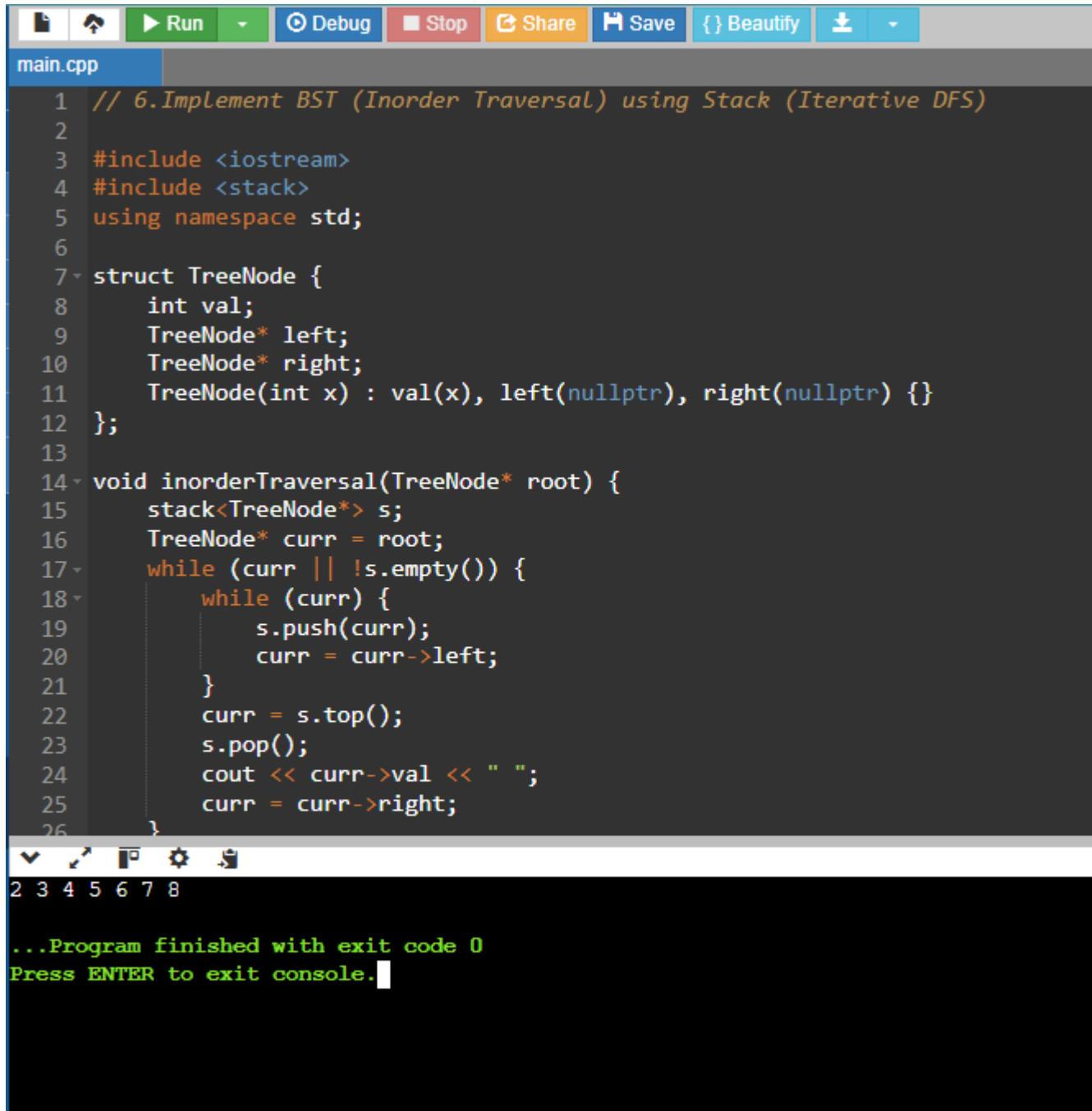


```
1 // 5.Implement Priority Queue using Stack
2
3 #include <iostream>
4 #include <stack>
5 using namespace std;
6
7 class PriorityQueue {
8     stack<int> s;
9
10 public:
11     void push(int x) {
12         stack<int> temp;
13         while (!s.empty() && s.top() > x) {
14             temp.push(s.top());
15             s.pop();
16         }
17         s.push(x);
18         while (!temp.empty()) {
19             s.push(temp.top());
20             temp.pop();
21         }
22     }
23
24     void pop() {
25         if (!s.empty()) s.pop();
26     }
27 }
```

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)



```
main.cpp
1 // 6.Implement BST (Inorder Traversal) using Stack (Iterative DFS)
2
3 #include <iostream>
4 #include <stack>
5 using namespace std;
6
7 struct TreeNode {
8     int val;
9     TreeNode* left;
10    TreeNode* right;
11    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12 };
13
14 void inorderTraversal(TreeNode* root) {
15     stack<TreeNode*> s;
16     TreeNode* curr = root;
17     while (curr || !s.empty()) {
18         while (curr) {
19             s.push(curr);
20             curr = curr->left;
21         }
22         curr = s.top();
23         s.pop();
24         cout << curr->val << " ";
25         curr = curr->right;
26     }
27 }
```

2 3 4 5 6 7 8

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

8. Implement Stack using Queue

The screenshot shows the LeetCode interface for the problem "Implement Stack using Queue". The left sidebar contains the problem description, which states: "All the calls to pop and top are valid." and a follow-up question: "Can you implement the stack using only one queue?". Below the description are statistics: 849.7K accepted, 1.3M submissions, and a 66.5% acceptance rate. The right pane shows the C++ code for a solution using two queues, q1 and q2. The code implements push, pop, top, and empty methods. The bottom right pane shows the test result, which is "Accepted" with a runtime of 0 ms. The input for the test case is ["MyStack", "push", "push", "top", "pop", "empty"] and the output is [[], [1], [2], [1], [], []].

```
1 class MyStack {
2     queue<int> q1, q2;
3
4 public:
5     MyStack() {}
6
7     void push(int x) {
8         q2.push(x);
9         while (!q1.empty()) {
10             q2.push(q1.front());
11             q1.pop();
12         }
13         swap(q1, q2);
14     }
15 }
```

9. Implement Deque using Queue

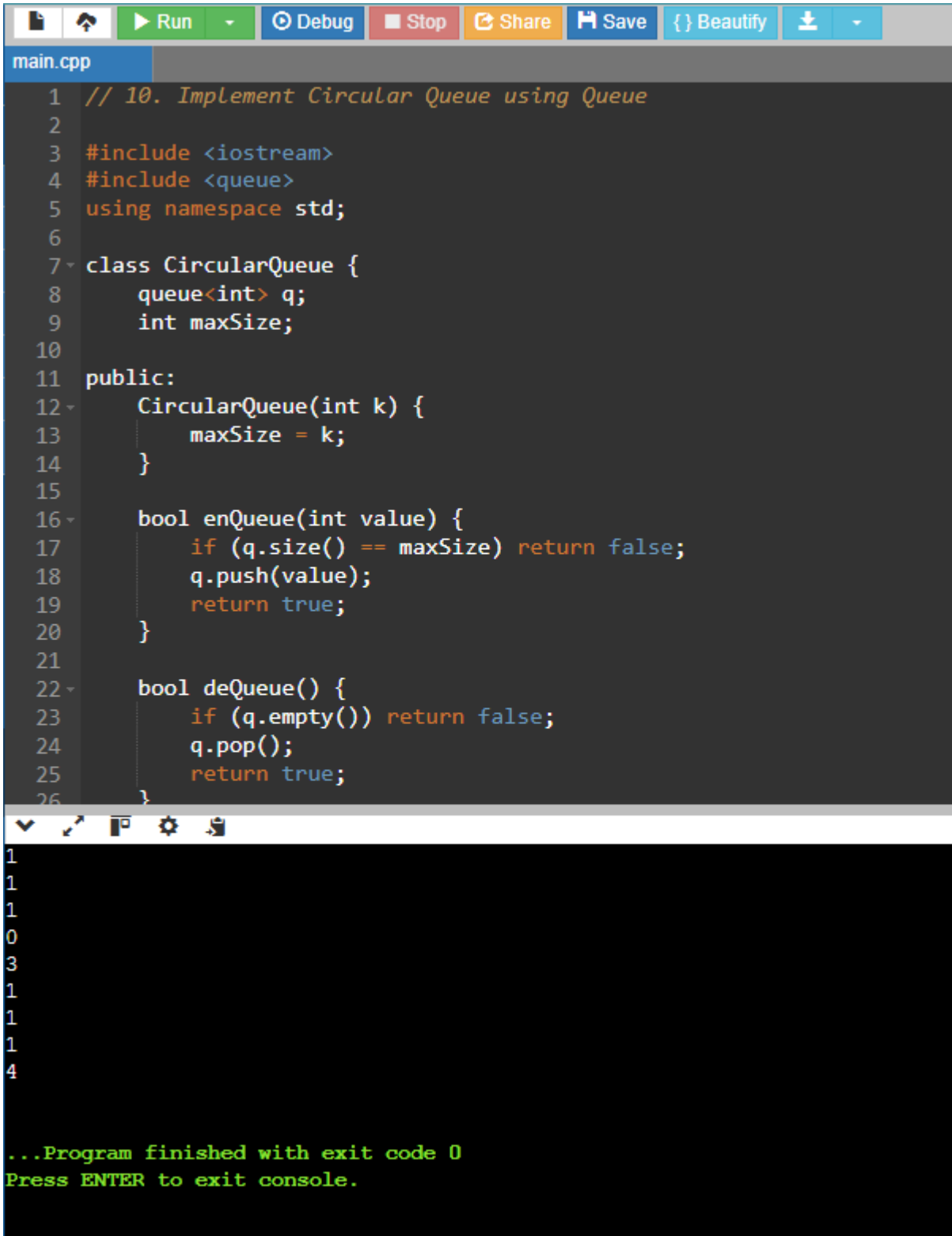
The screenshot shows a C++ program in a code editor. The program implements a Deque using two queues, q1 and q2. The code includes headers for iostream and queue, and uses the std namespace. The Deque class has methods for insertFront, insertRear, transfer, and pop. The main function tests the Deque by inserting 5 at the front and 25 at the rear, then deleting the front element and printing the state. The output shows the front and rear elements after deletion.

```
1 // 9. Implement Deque using Queue
2
3 #include <iostream>
4 #include <queue>
5 using namespace std;
6
7 class Deque {
8     queue<int> q1, q2;
9
10    void transfer(queue<int>& src, queue<int>& dest) {
11        while (!src.empty()) {
12            dest.push(src.front());
13            src.pop();
14        }
15    }
16
17 public:
18    void insertFront(int x) {
19        q2.push(x);
20        transfer(q1, q2);
21        swap(q1, q2);
22    }
23
24    void insertRear(int x) {
25        q1.push(x);
26    }
27 }
```

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

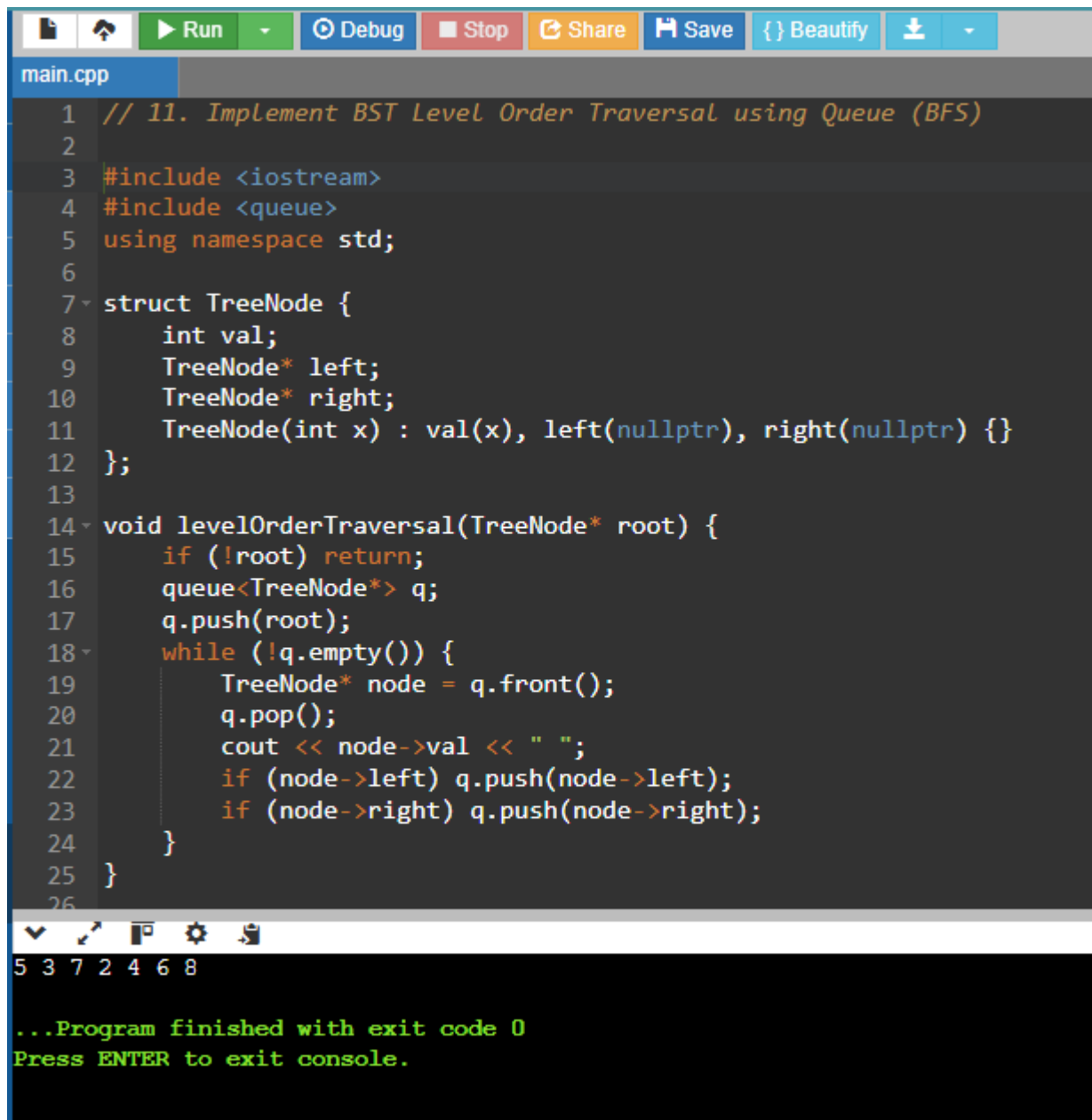


```
main.cpp
1 // 10. Implement Circular Queue using Queue
2
3 #include <iostream>
4 #include <queue>
5 using namespace std;
6
7 class CircularQueue {
8     queue<int> q;
9     int maxSize;
10
11 public:
12     CircularQueue(int k) {
13         maxSize = k;
14     }
15
16     bool enqueue(int value) {
17         if (q.size() == maxSize) return false;
18         q.push(value);
19         return true;
20     }
21
22     bool dequeue() {
23         if (q.empty()) return false;
24         q.pop();
25         return true;
26     }
27 }
```

1
1
1
1
0
3
1
1
1
1
4

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

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

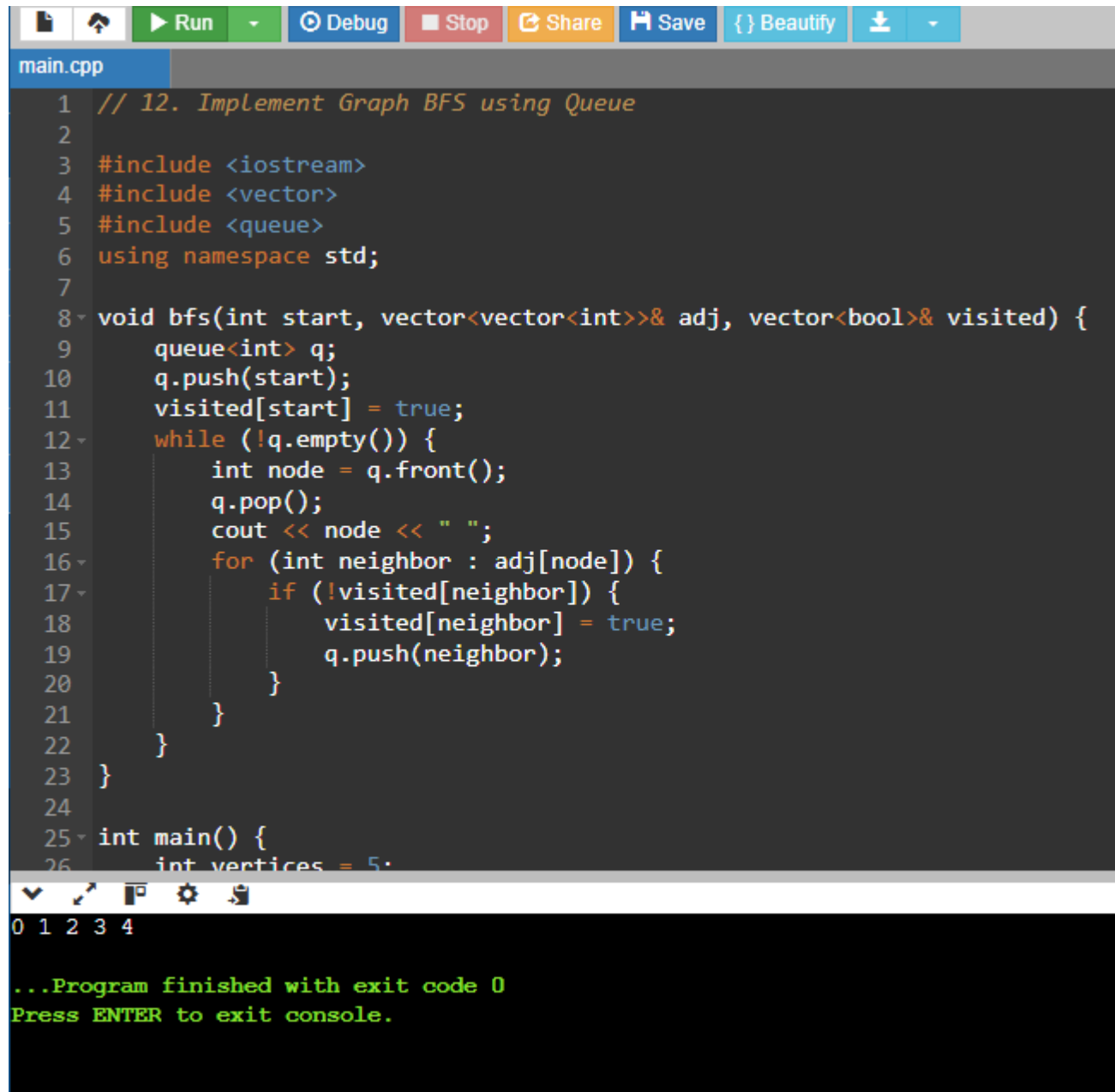


```
main.cpp
1 // 11. Implement BST Level Order Traversal using Queue (BFS)
2
3 #include <iostream>
4 #include <queue>
5 using namespace std;
6
7 struct TreeNode {
8     int val;
9     TreeNode* left;
10    TreeNode* right;
11    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12 };
13
14 void levelOrderTraversal(TreeNode* root) {
15     if (!root) return;
16     queue<TreeNode*> q;
17     q.push(root);
18     while (!q.empty()) {
19         TreeNode* node = q.front();
20         q.pop();
21         cout << node->val << " ";
22         if (node->left) q.push(node->left);
23         if (node->right) q.push(node->right);
24     }
25 }
26
```

5 3 7 2 4 6 8

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

12. Implement Graph BFS using Queue

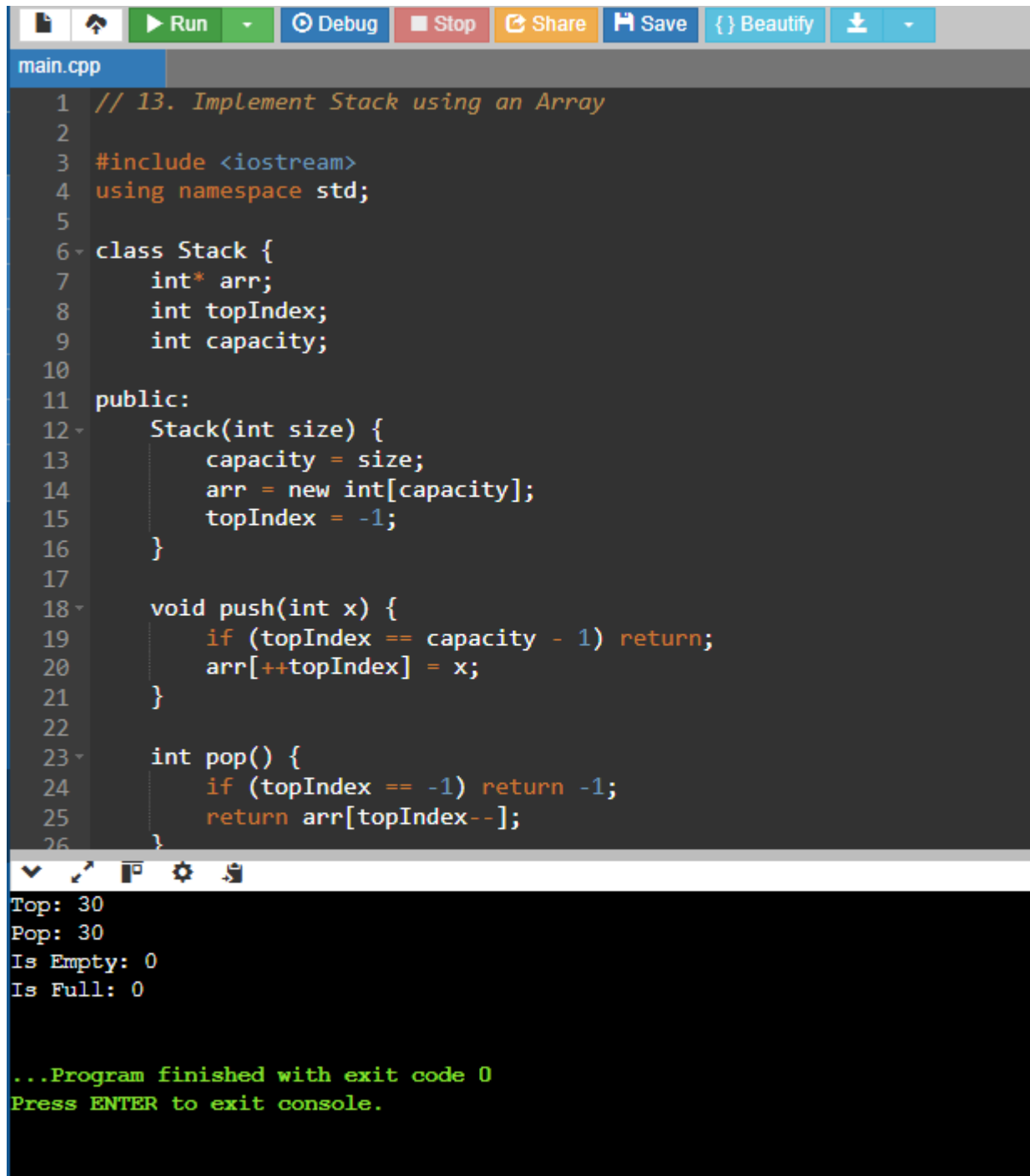


```
main.cpp
1 // 12. Implement Graph BFS using Queue
2
3 #include <iostream>
4 #include <vector>
5 #include <queue>
6 using namespace std;
7
8 void bfs(int start, vector<vector<int>>& adj, vector<bool>& visited) {
9     queue<int> q;
10    q.push(start);
11    visited[start] = true;
12    while (!q.empty()) {
13        int node = q.front();
14        q.pop();
15        cout << node << " ";
16        for (int neighbor : adj[node]) {
17            if (!visited[neighbor]) {
18                visited[neighbor] = true;
19                q.push(neighbor);
20            }
21        }
22    }
23 }
24
25 int main() {
26     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



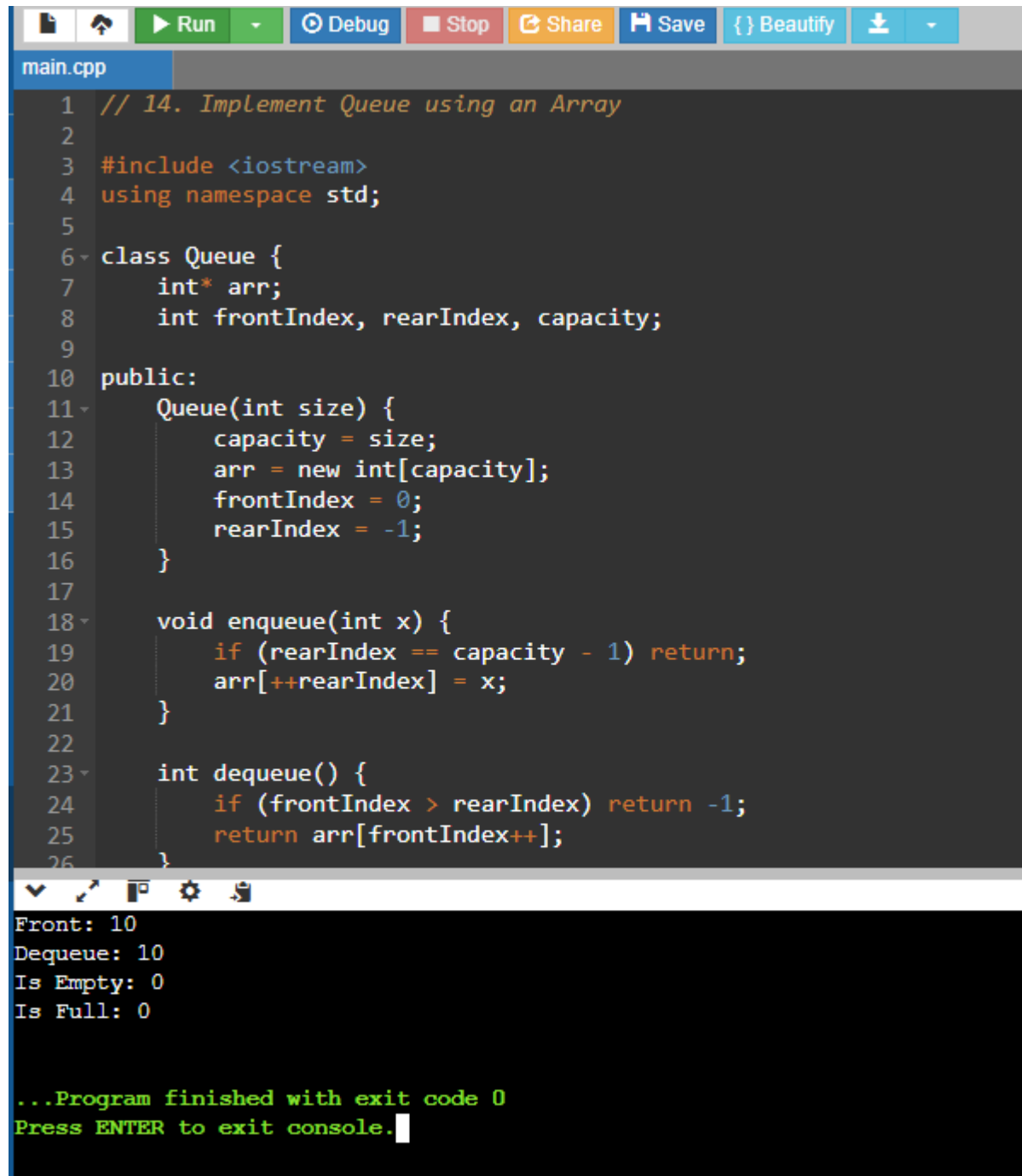
The image shows a C++ IDE with a file named `main.cpp`. The code implements a `Stack` class using an array. The class has three private attributes: `int* arr`, `int topIndex`, and `int capacity`. It has three public methods: `Stack(int size)` for initialization, `push(int x)` for adding elements, and `pop()` for removing elements. The `push` method checks if the stack is full before adding an element. The `pop` method checks if the stack is empty before removing an element. The IDE's output window shows the results of running the program: `Top: 30`, `Pop: 30`, `Is Empty: 0`, and `Is Full: 0`. The program finished with exit code 0.

```
1 // 13. Implement Stack using an Array
2
3 #include <iostream>
4 using namespace std;
5
6 class Stack {
7     int* arr;
8     int topIndex;
9     int capacity;
10
11 public:
12     Stack(int size) {
13         capacity = size;
14         arr = new int[capacity];
15         topIndex = -1;
16     }
17
18     void push(int x) {
19         if (topIndex == capacity - 1) return;
20         arr[++topIndex] = x;
21     }
22
23     int pop() {
24         if (topIndex == -1) return -1;
25         return arr[topIndex--];
26     }
27 }
```

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



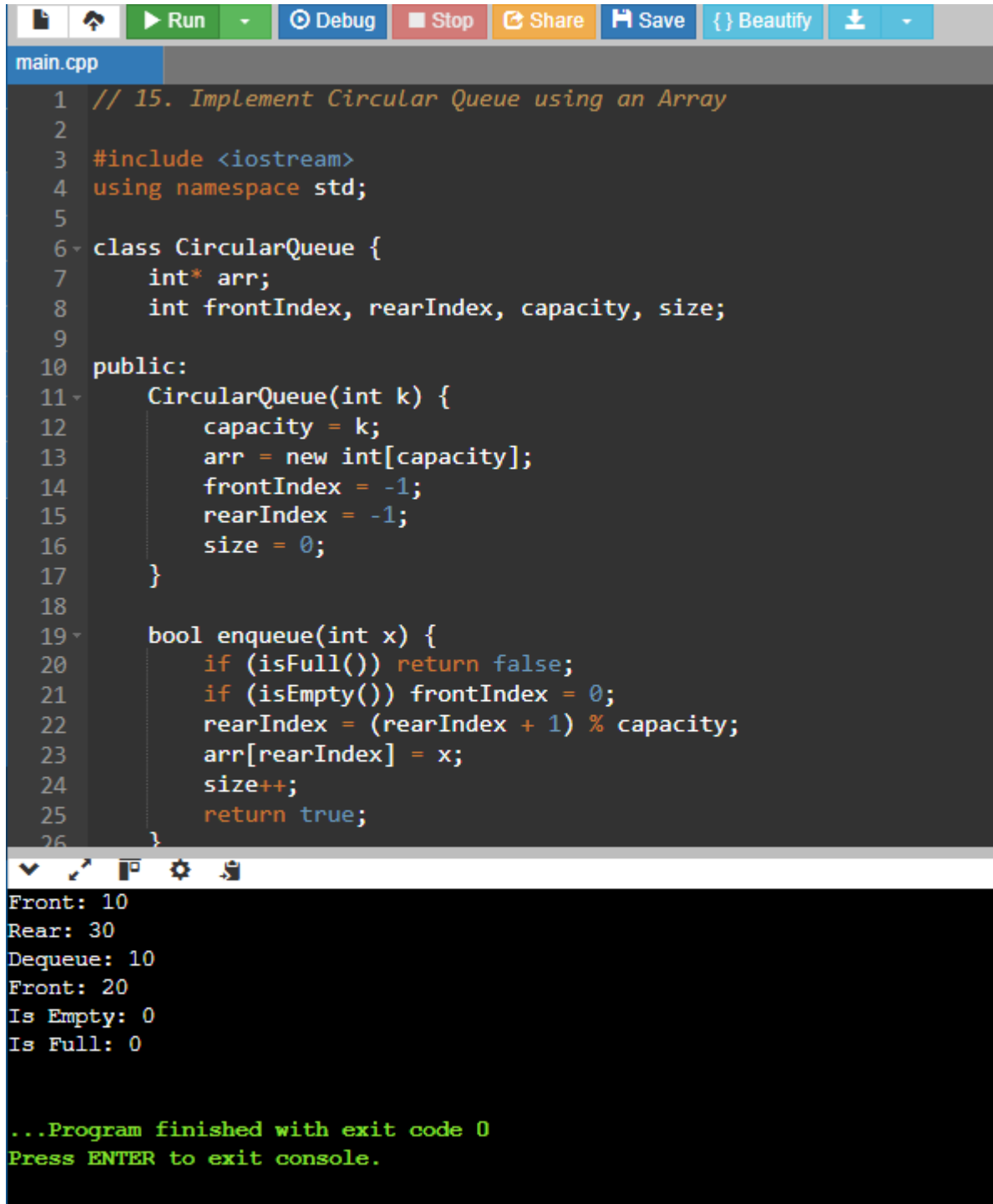
The image shows a C++ IDE window titled "main.cpp". The code implements a queue using an array. The class `Queue` has attributes `arr`, `frontIndex`, `rearIndex`, and `capacity`. It includes methods `enqueue` and `dequeue`. The `enqueue` method checks if the queue is full (if `rearIndex == capacity - 1`) and returns if so; otherwise, it increments `rearIndex` and stores the value `x` in `arr`. The `dequeue` method checks if the queue is empty (if `frontIndex > rearIndex`) and returns -1 if so; otherwise, it increments `frontIndex` and returns the value from `arr`. The output window shows the results of running the program: `Front: 10`, `Dequeue: 10`, `Is Empty: 0`, and `Is Full: 0`. The program finished with exit code 0, and the user is prompted to press ENTER to exit the console.

```
1 // 14. Implement Queue using an Array
2
3 #include <iostream>
4 using namespace std;
5
6 class Queue {
7     int* arr;
8     int frontIndex, rearIndex, capacity;
9
10 public:
11     Queue(int size) {
12         capacity = size;
13         arr = new int[capacity];
14         frontIndex = 0;
15         rearIndex = -1;
16     }
17
18     void enqueue(int x) {
19         if (rearIndex == capacity - 1) return;
20         arr[++rearIndex] = x;
21     }
22
23     int dequeue() {
24         if (frontIndex > rearIndex) return -1;
25         return arr[frontIndex++];
26     }
27 }
```

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

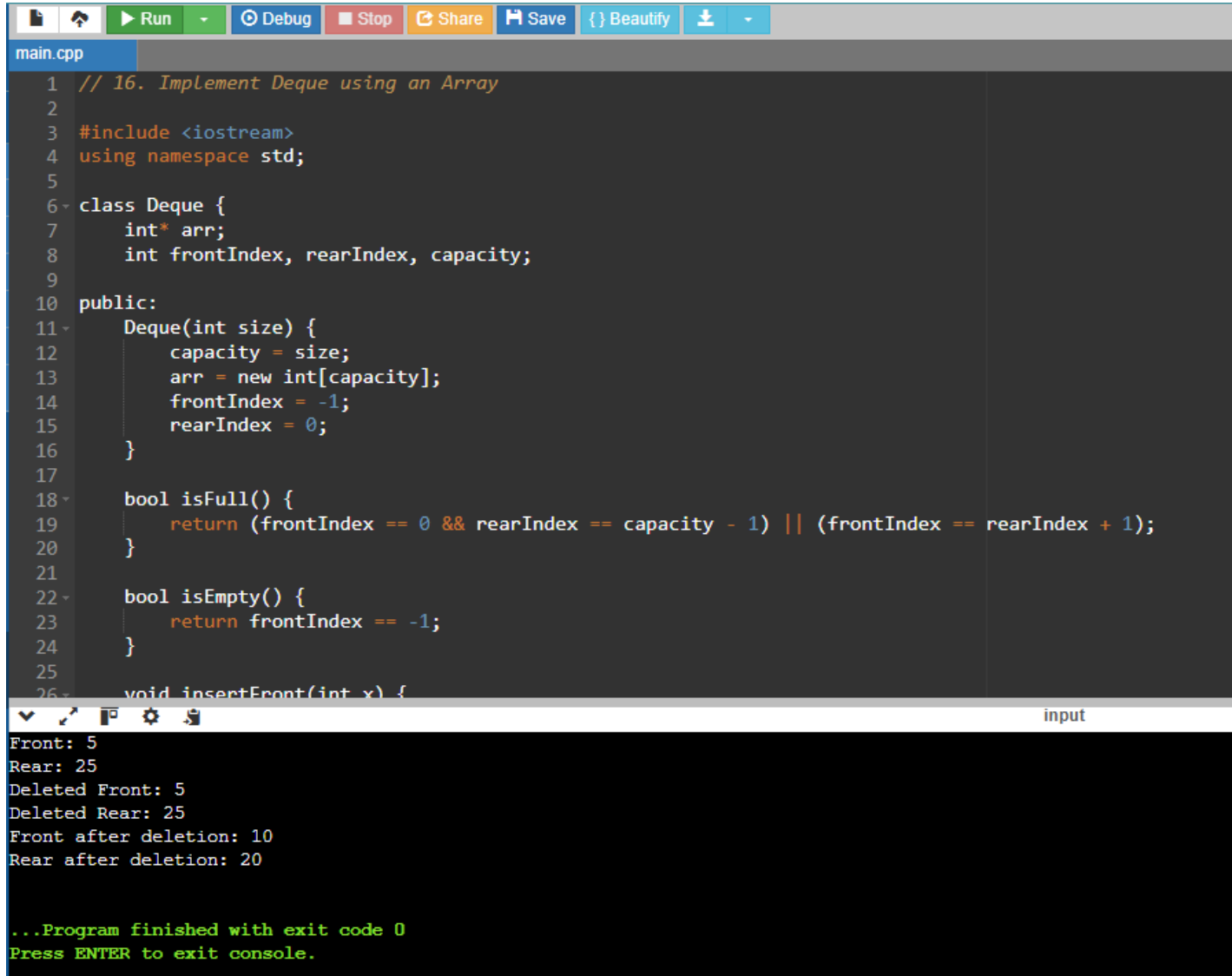


```
1 // 15. Implement Circular Queue using an Array
2
3 #include <iostream>
4 using namespace std;
5
6 class CircularQueue {
7     int* arr;
8     int frontIndex, rearIndex, capacity, size;
9
10 public:
11     CircularQueue(int k) {
12         capacity = k;
13         arr = new int[capacity];
14         frontIndex = -1;
15         rearIndex = -1;
16         size = 0;
17     }
18
19     bool enqueue(int x) {
20         if (isFull()) return false;
21         if (isEmpty()) frontIndex = 0;
22         rearIndex = (rearIndex + 1) % capacity;
23         arr[rearIndex] = x;
24         size++;
25         return true;
26     }
27
28     bool dequeue() {
29         if (isEmpty()) return false;
30         frontIndex = (frontIndex + 1) % capacity;
31         size--;
32         return true;
33     }
34
35     bool isFull() {
36         return (frontIndex == rearIndex);
37     }
38
39     bool isEmpty() {
40         return (frontIndex == -1);
41     }
42
43     int getFront() {
44         return frontIndex;
45     }
46
47     int getRear() {
48         return rearIndex;
49     }
50
51     int getSize() {
52         return size;
53     }
54
55     int* getArray() {
56         return arr;
57     }
58
59     void display() {
60         cout << "Front: " << frontIndex << endl;
61         cout << "Rear: " << rearIndex << endl;
62         cout << "Dequeue: " << frontIndex << endl;
63         cout << "Front: " << frontIndex << endl;
64         cout << "Is Empty: " << isEmpty() << endl;
65         cout << "Is Full: " << isFull() << endl;
66     }
67
68     ~CircularQueue() {
69         delete arr;
70     }
71 }
```

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



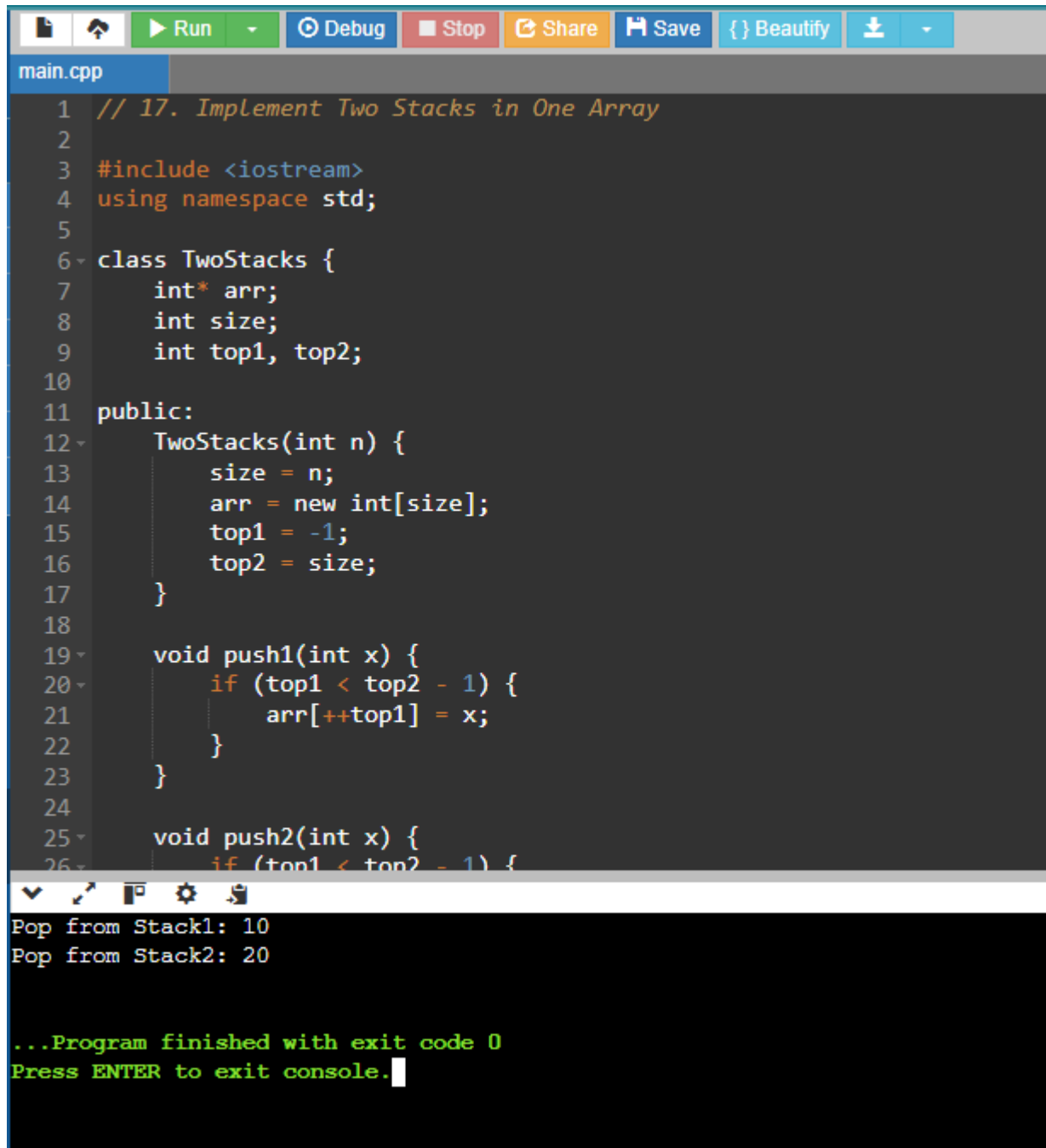
```
1 // 16. Implement Deque using an Array
2
3 #include <iostream>
4 using namespace std;
5
6 class Deque {
7     int* arr;
8     int frontIndex, rearIndex, capacity;
9
10 public:
11     Deque(int size) {
12         capacity = size;
13         arr = new int[capacity];
14         frontIndex = -1;
15         rearIndex = 0;
16     }
17
18     bool isFull() {
19         return (frontIndex == 0 && rearIndex == capacity - 1) || (frontIndex == rearIndex + 1);
20     }
21
22     bool isEmpty() {
23         return frontIndex == -1;
24     }
25
26     void insertFront(int x) {
```

input

```
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



```
main.cpp
1 // 17. Implement Two Stacks in One Array
2
3 #include <iostream>
4 using namespace std;
5
6 class TwoStacks {
7     int* arr;
8     int size;
9     int top1, top2;
10
11 public:
12     TwoStacks(int n) {
13         size = n;
14         arr = new int[size];
15         top1 = -1;
16         top2 = size;
17     }
18
19     void push1(int x) {
20         if (top1 < top2 - 1) {
21             arr[++top1] = x;
22         }
23     }
24
25     void push2(int x) {
26         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

Run

Debug

Stop

Share

Save

{ } Beautify

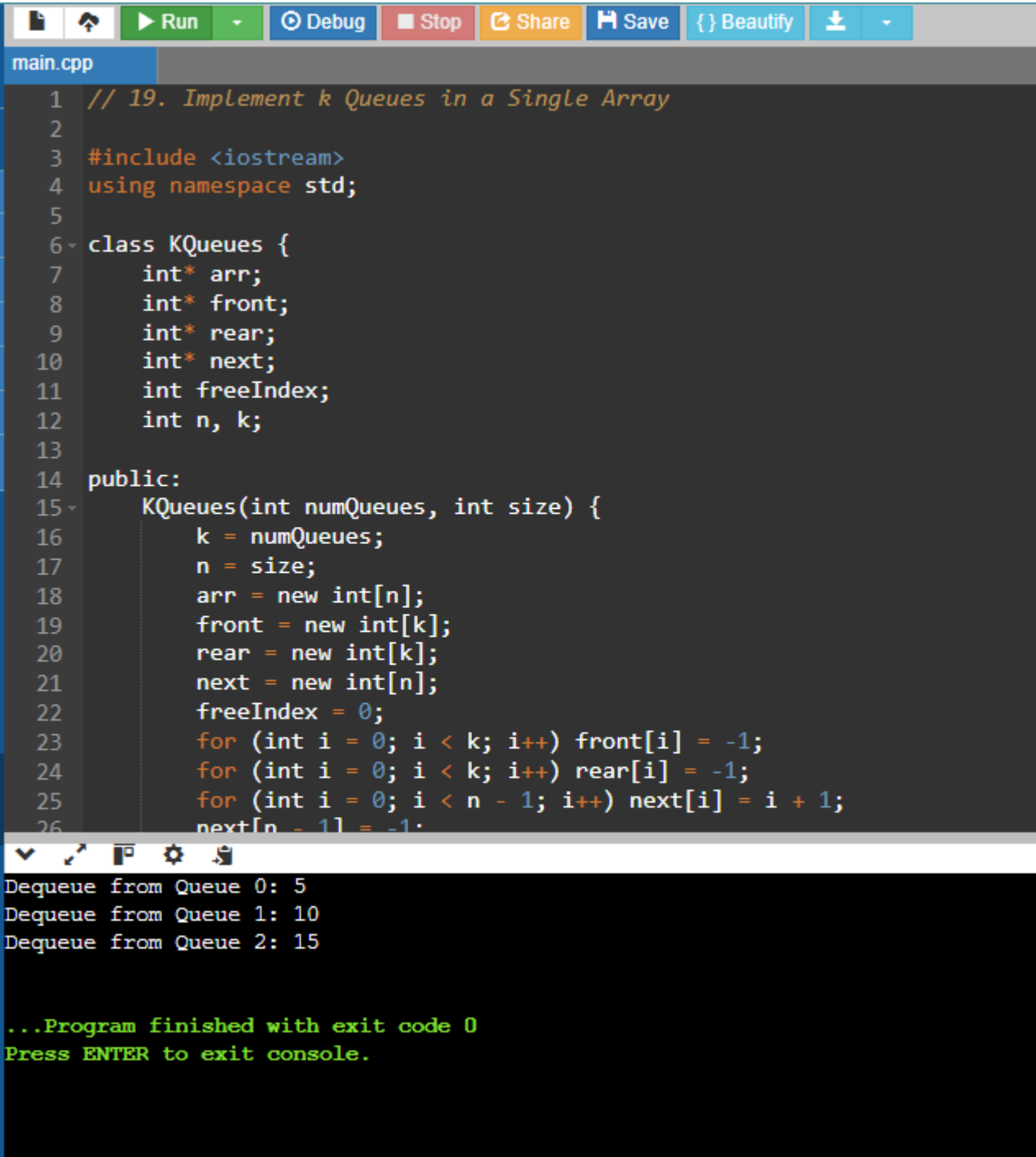
main.cpp

```
1 // 18. Implement k Stacks in a Single Array
2
3 #include <iostream>
4 #include <vector>
5 using namespace std;
6
7 class KStacks {
8     int* arr;
9     int* top;
10    int* next;
11    int freeIndex;
12    int n, k;
13
14 public:
15     KStacks(int numStacks, int size) {
16         k = numStacks;
17         n = size;
18         arr = new int[n];
19         top = new int[k];
20         next = new int[n];
21         freeIndex = 0;
22         for (int i = 0; i < k; i++) top[i] = -1;
23         for (int i = 0; i < n - 1; i++) next[i] = i + 1;
24         next[n - 1] = -1;
25     }
26 }
```

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

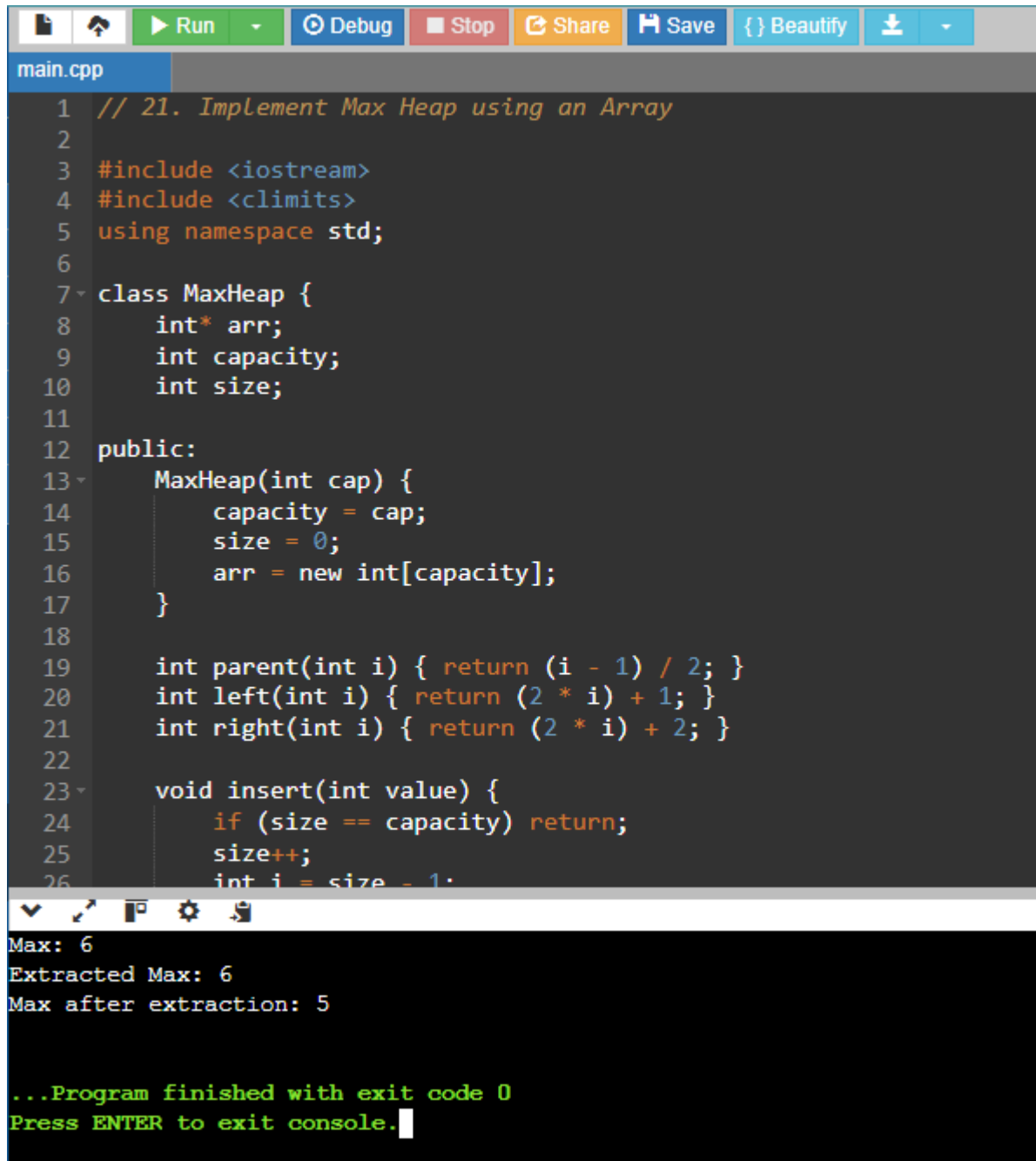


```
1 // 19. Implement k Queues in a Single Array
2
3 #include <iostream>
4 using namespace std;
5
6 class KQueues {
7     int* arr;
8     int* front;
9     int* rear;
10    int* next;
11    int freeIndex;
12    int n, k;
13
14 public:
15     KQueues(int numQueues, int size) {
16         k = numQueues;
17         n = size;
18         arr = new int[n];
19         front = new int[k];
20         rear = new int[k];
21         next = new int[n];
22         freeIndex = 0;
23         for (int i = 0; i < k; i++) front[i] = -1;
24         for (int i = 0; i < k; i++) rear[i] = -1;
25         for (int i = 0; i < n - 1; i++) next[i] = i + 1;
26         next[n - 1] = -1;
27     }
28
29     void enqueue(int value) {
30         // Implementation of enqueue
31     }
32
33     int dequeue(int queueIndex) {
34         // Implementation of dequeue
35     }
36
37     void display() {
38         // Implementation of display
39     }
40
41     ~KQueues() {
42         delete arr;
43         delete front;
44         delete rear;
45         delete next;
46     }
47 };
48
49 int main() {
50     KQueues kq(3, 15);
51     kq.enqueue(5);
52     kq.enqueue(10);
53     kq.enqueue(15);
54     kq.dequeue(0);
55     kq.dequeue(1);
56     kq.dequeue(2);
57     kq.display();
58     return 0;
59 }
```

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



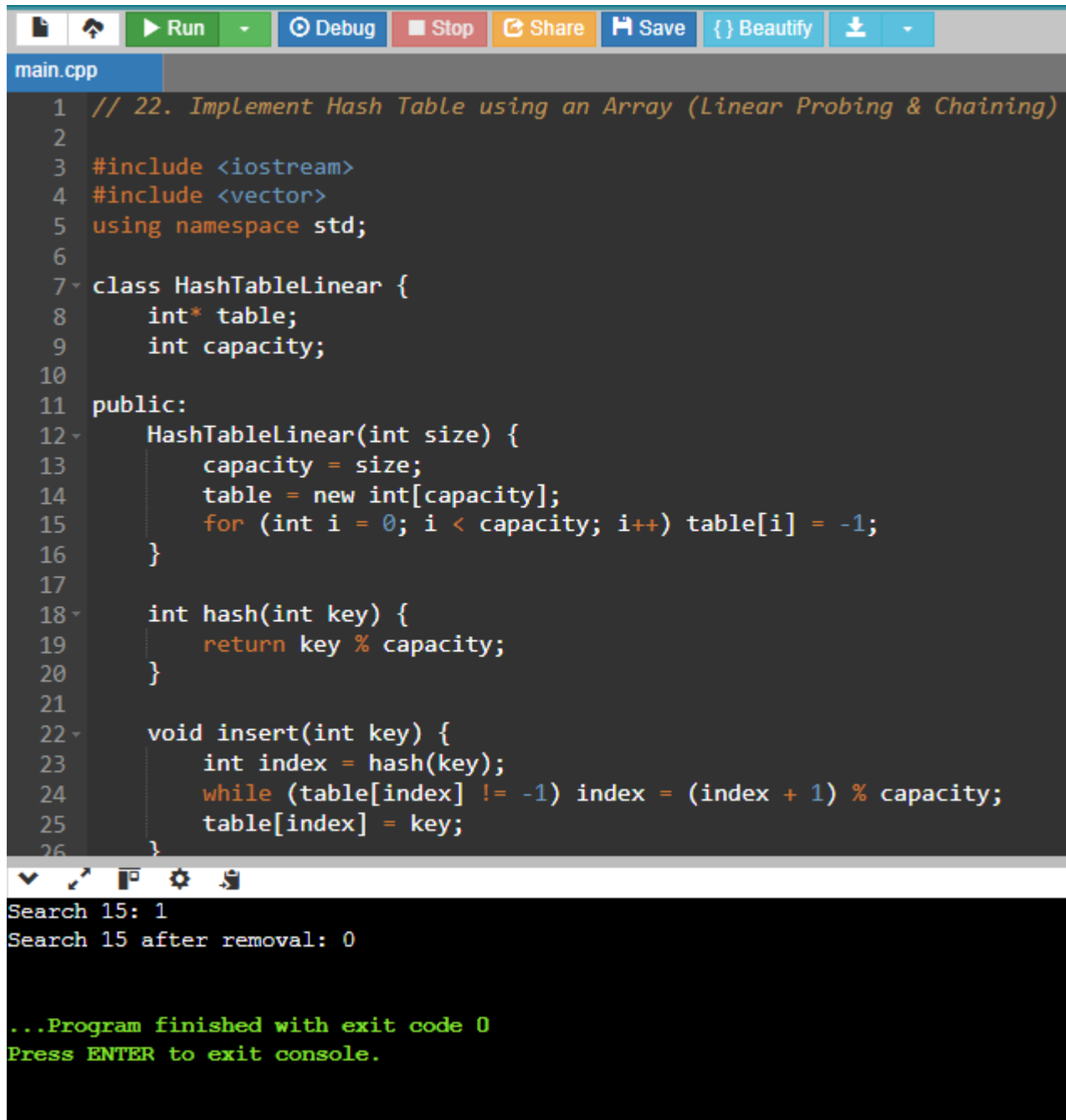
The image shows a C++ IDE with a file named `main.cpp`. The code implements a `MaxHeap` class using an array. The class has attributes `arr`, `capacity`, and `size`. It includes methods for `parent`, `left`, `right`, and `insert`. The `insert` method checks if the heap is full, increments the size, and starts inserting the new value at the end of the array. The output window shows the execution results: `Max: 6`, `Extracted Max: 6`, and `Max after extraction: 5`. The program finishes with exit code 0.

```
1 // 21. Implement Max Heap using an Array
2
3 #include <iostream>
4 #include <climits>
5 using namespace std;
6
7 class MaxHeap {
8     int* arr;
9     int capacity;
10    int size;
11
12 public:
13    MaxHeap(int cap) {
14        capacity = cap;
15        size = 0;
16        arr = new int[capacity];
17    }
18
19    int parent(int i) { return (i - 1) / 2; }
20    int left(int i) { return (2 * i) + 1; }
21    int right(int i) { return (2 * i) + 2; }
22
23    void insert(int value) {
24        if (size == capacity) return;
25        size++;
26        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)

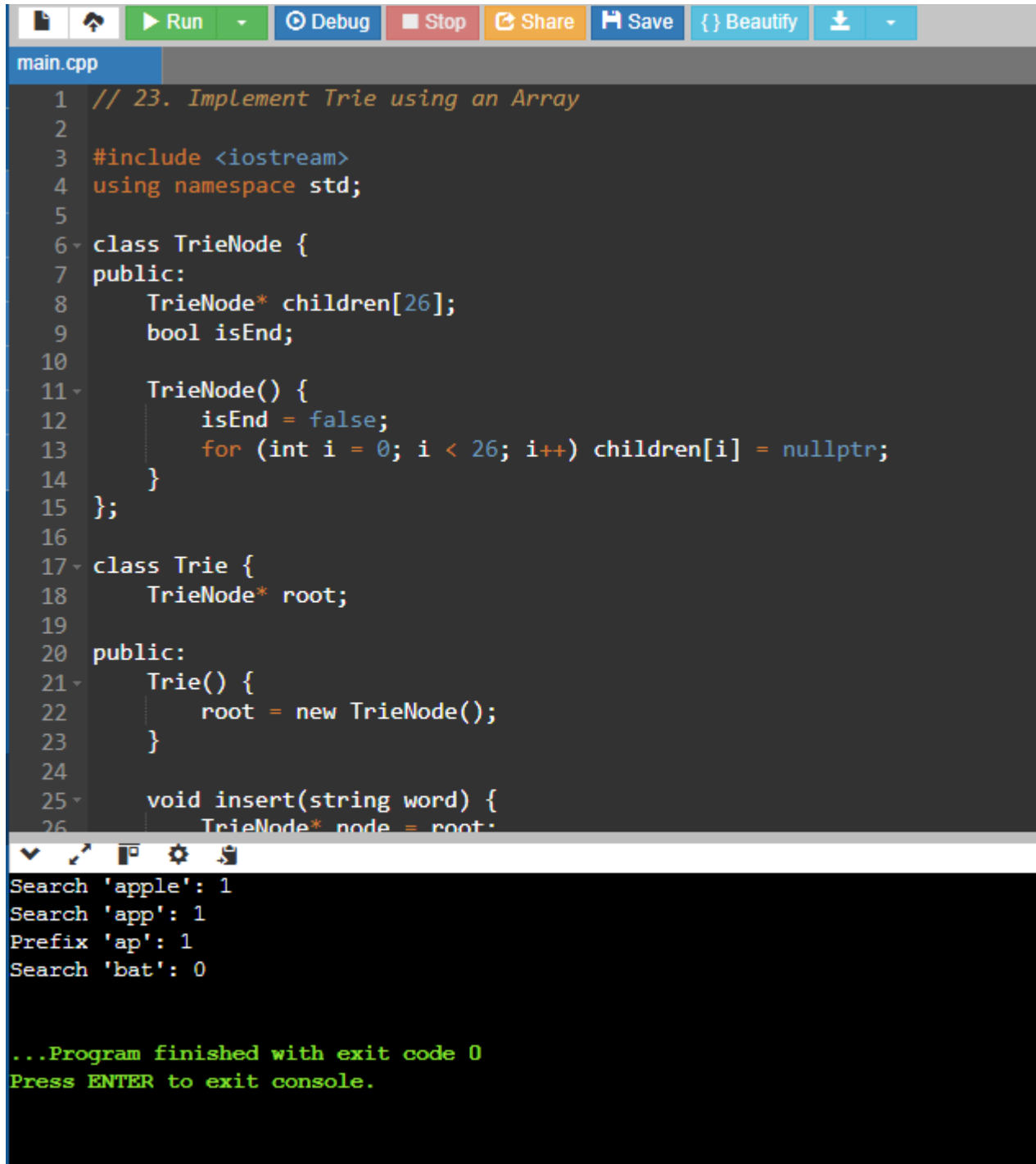


```
main.cpp
1 // 22. Implement Hash Table using an Array (Linear Probing & Chaining)
2
3 #include <iostream>
4 #include <vector>
5 using namespace std;
6
7 class HashTableLinear {
8     int* table;
9     int capacity;
10
11 public:
12     HashTableLinear(int size) {
13         capacity = size;
14         table = new int[capacity];
15         for (int i = 0; i < capacity; i++) table[i] = -1;
16     }
17
18     int hash(int key) {
19         return key % capacity;
20     }
21
22     void insert(int key) {
23         int index = hash(key);
24         while (table[index] != -1) index = (index + 1) % capacity;
25         table[index] = key;
26     }
27
28     int search(int key) {
29         int index = hash(key);
30         while (table[index] != -1) {
31             if (table[index] == key) return index;
32             index = (index + 1) % capacity;
33         }
34         return -1;
35     }
36
37     void remove(int key) {
38         int index = search(key);
39         if (index != -1) table[index] = -1;
40     }
41
42     void display() {
43         for (int i = 0; i < capacity; i++) {
44             cout << table[i] << " ";
45             if (i % 10 == 9) cout << endl;
46         }
47     }
48 };
49
50 int main() {
51     HashTableLinear ht(10);
52     ht.insert(15);
53     ht.insert(20);
54     ht.insert(30);
55     ht.insert(40);
56     ht.insert(50);
57     ht.insert(60);
58     ht.insert(70);
59     ht.insert(80);
60     ht.insert(90);
61     ht.insert(100);
62     ht.display();
63     int key = 15;
64     int index = ht.search(key);
65     cout << "Search " << key << ": " << index << endl;
66     ht.remove(key);
67     index = ht.search(key);
68     cout << "Search " << key << " after removal: " << index << endl;
69     return 0;
70 }
```

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

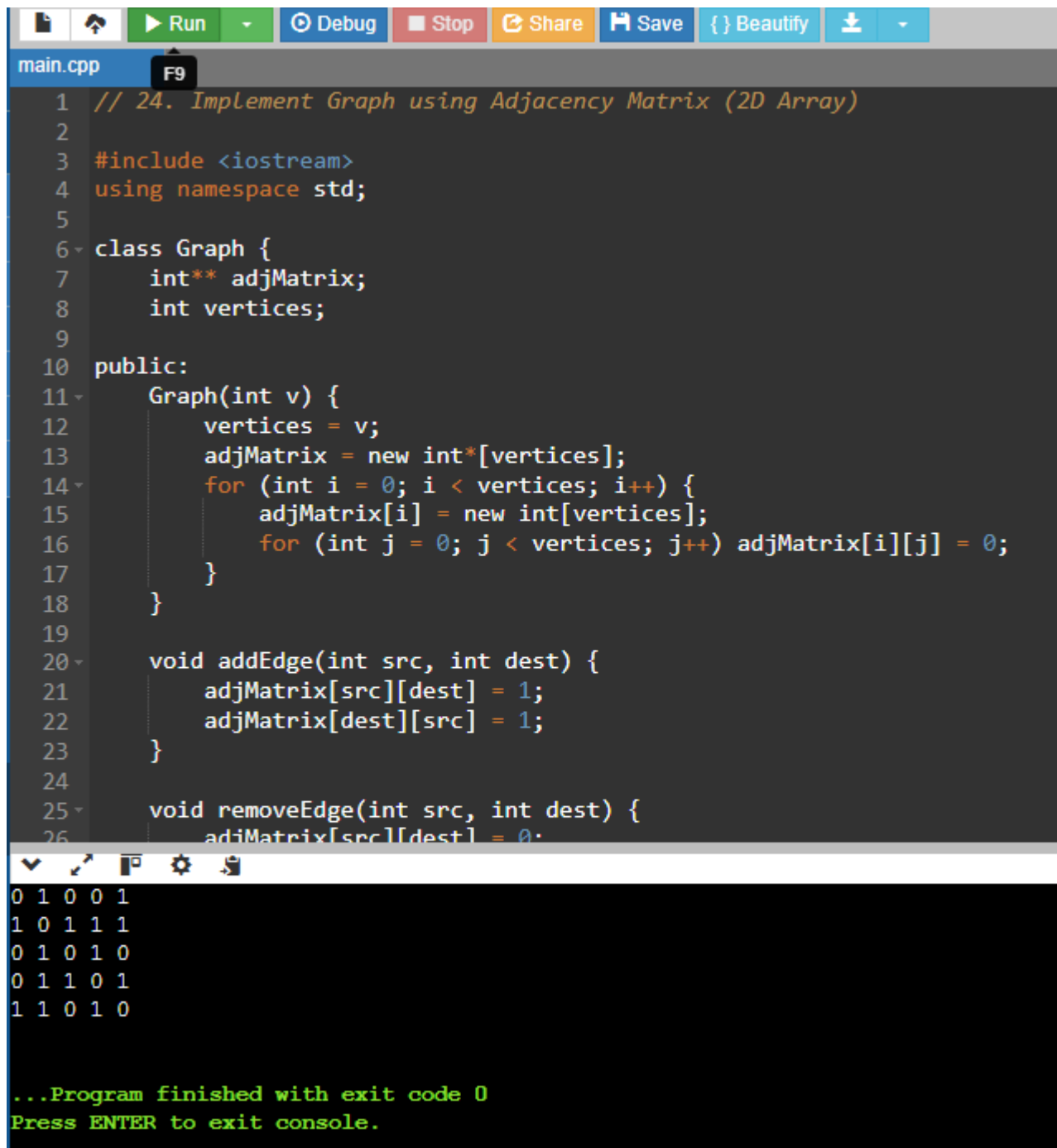


```
1 // 23. Implement Trie using an Array
2
3 #include <iostream>
4 using namespace std;
5
6 class TrieNode {
7 public:
8     TrieNode* children[26];
9     bool isEnd;
10
11     TrieNode() {
12         isEnd = false;
13         for (int i = 0; i < 26; i++) children[i] = nullptr;
14     }
15 };
16
17 class Trie {
18     TrieNode* root;
19
20 public:
21     Trie() {
22         root = new TrieNode();
23     }
24
25     void insert(string word) {
26         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)

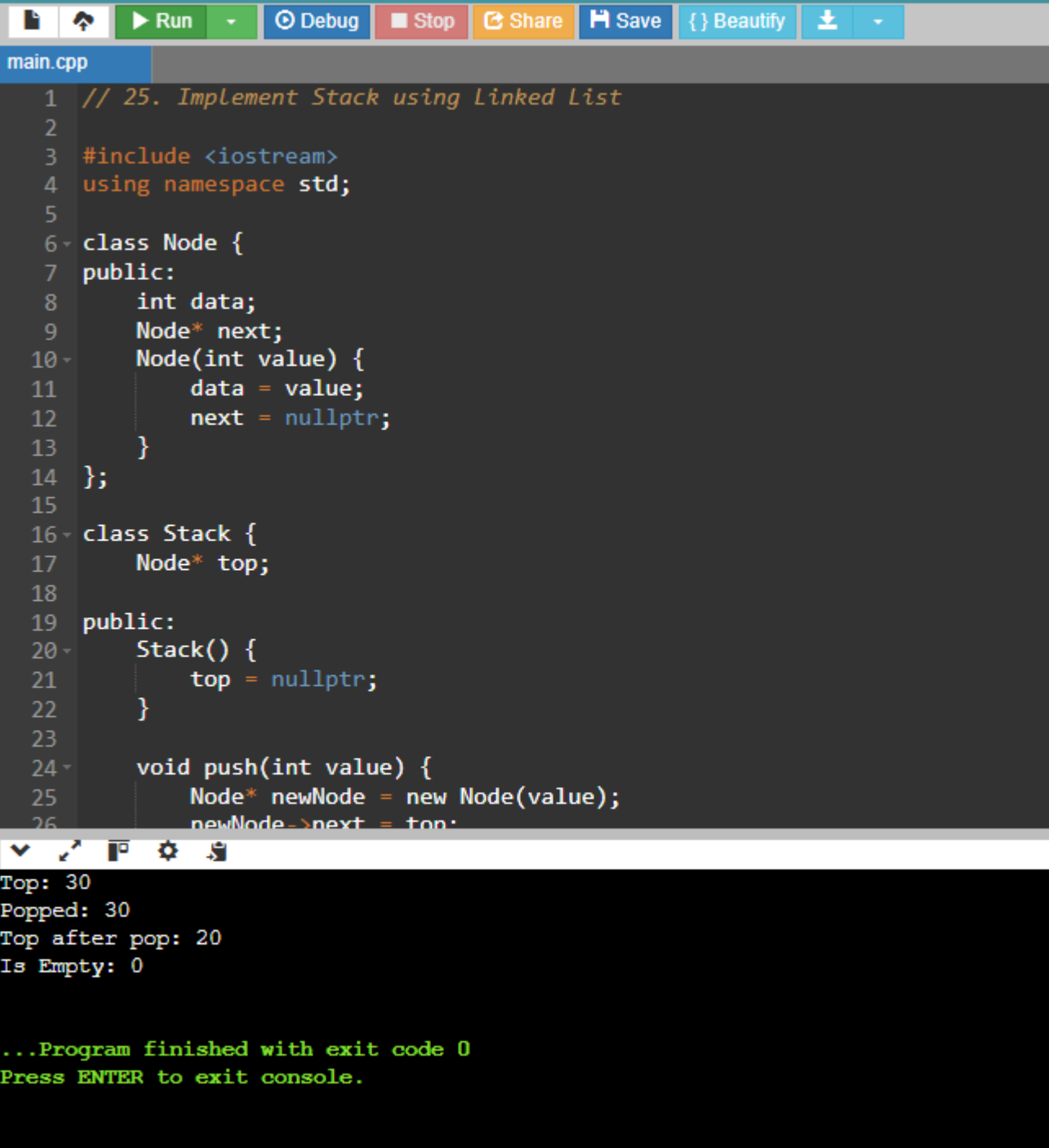


```
main.cpp F9
1 // 24. Implement Graph using Adjacency Matrix (2D Array)
2
3 #include <iostream>
4 using namespace std;
5
6 class Graph {
7     int** adjMatrix;
8     int vertices;
9
10 public:
11     Graph(int v) {
12         vertices = v;
13         adjMatrix = new int*[vertices];
14         for (int i = 0; i < vertices; i++) {
15             adjMatrix[i] = new int[vertices];
16             for (int j = 0; j < vertices; j++) adjMatrix[i][j] = 0;
17         }
18     }
19
20     void addEdge(int src, int dest) {
21         adjMatrix[src][dest] = 1;
22         adjMatrix[dest][src] = 1;
23     }
24
25     void removeEdge(int src, int dest) {
26         adjMatrix[src][dest] = 0;
27     }
28 }
29
30 int main() {
31     Graph g(5);
32     g.addEdge(0, 1);
33     g.addEdge(0, 3);
34     g.addEdge(1, 2);
35     g.addEdge(1, 4);
36     g.addEdge(2, 4);
37
38     for (int i = 0; i < 5; i++) {
39         for (int j = 0; j < 5; j++) {
40             cout << adjMatrix[i][j] << " ";
41         }
42         cout << endl;
43     }
44
45     return 0;
46 }
```

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

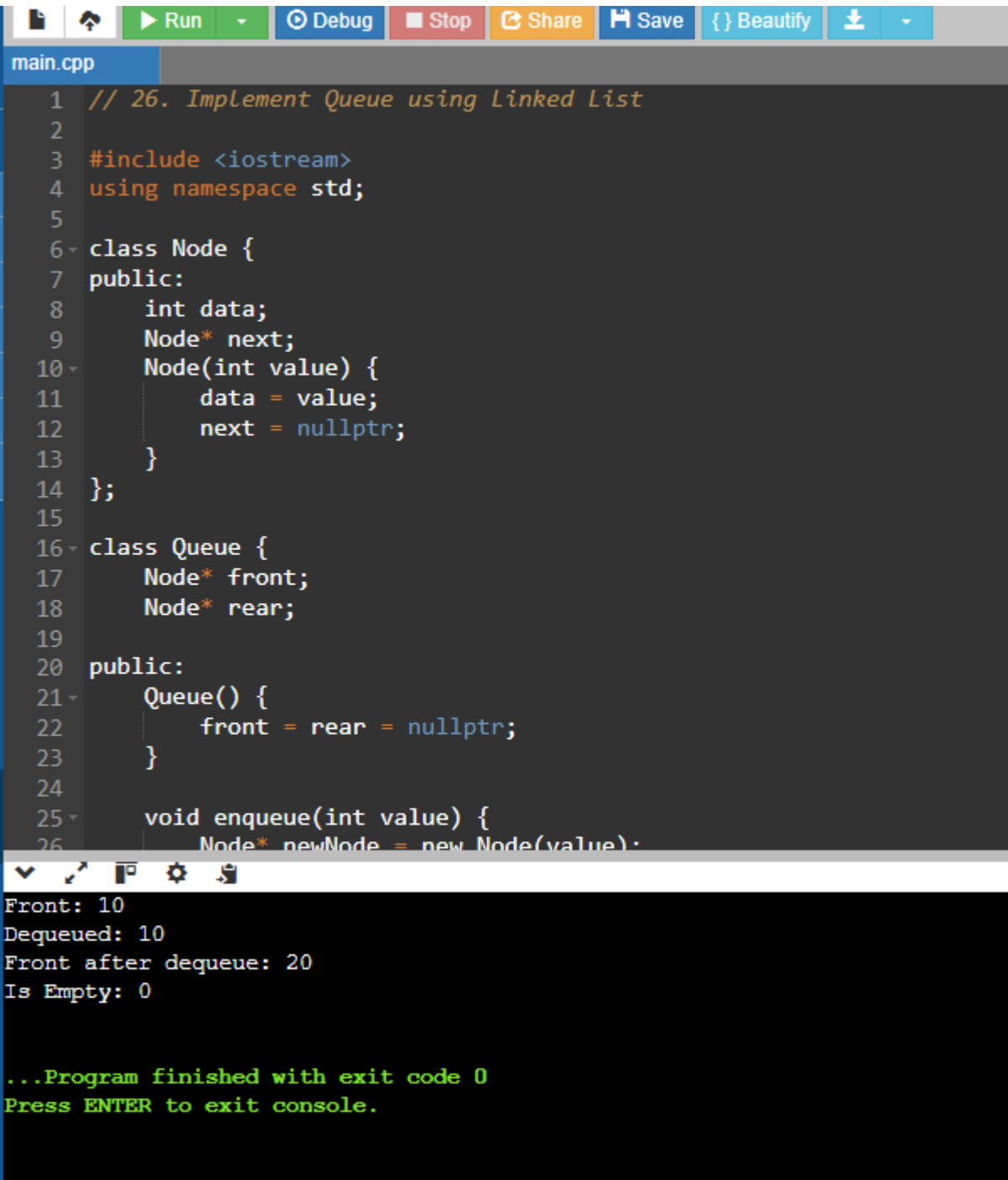


```
1 // 25. Implement Stack using Linked List
2
3 #include <iostream>
4 using namespace std;
5
6 class Node {
7 public:
8     int data;
9     Node* next;
10    Node(int value) {
11        data = value;
12        next = nullptr;
13    }
14 };
15
16 class Stack {
17     Node* top;
18
19 public:
20     Stack() {
21         top = nullptr;
22     }
23
24     void push(int value) {
25         Node* newNode = new Node(value);
26         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

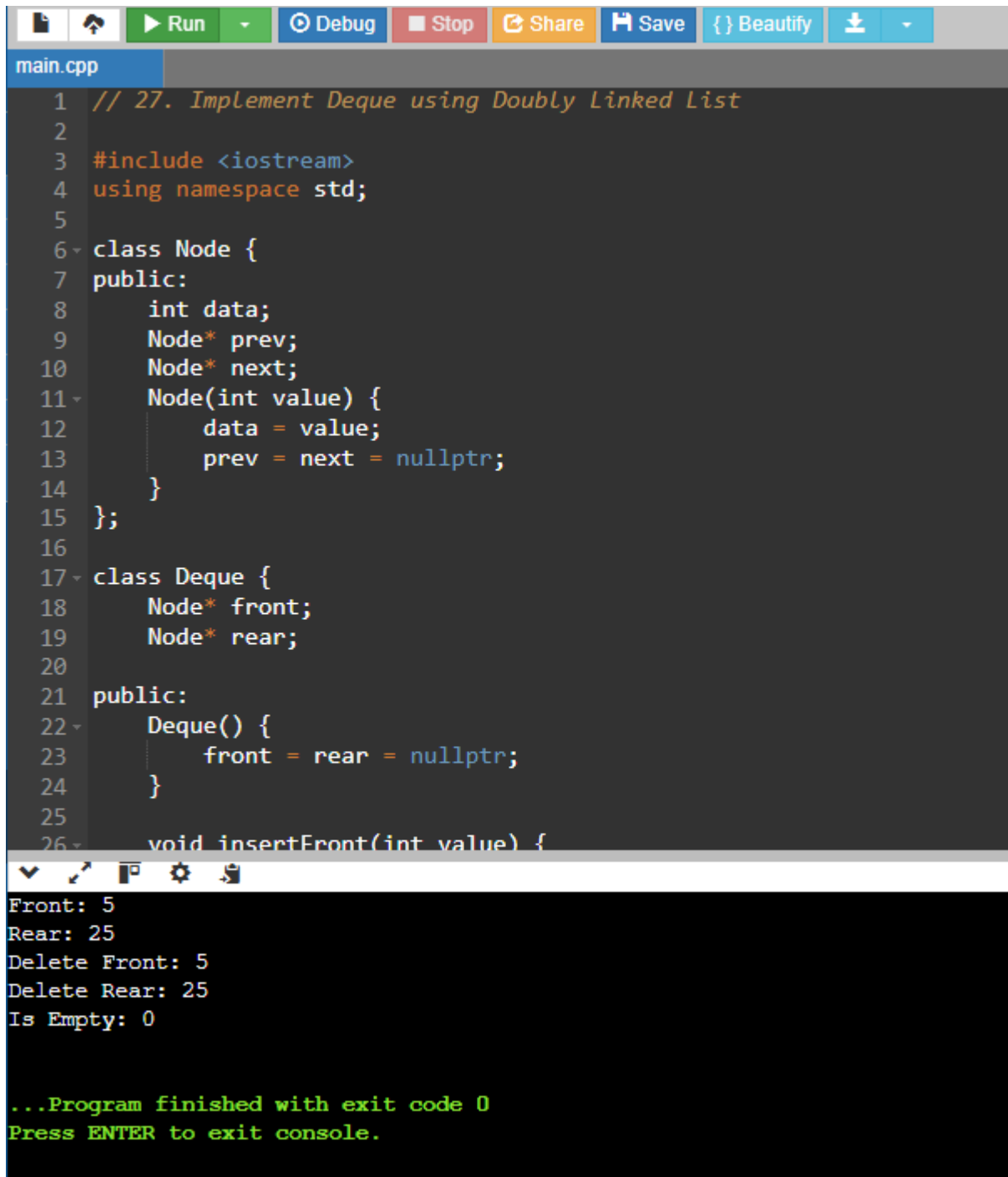


```
main.cpp
1 // 26. Implement Queue using Linked List
2
3 #include <iostream>
4 using namespace std;
5
6 class Node {
7 public:
8     int data;
9     Node* next;
10    Node(int value) {
11        data = value;
12        next = nullptr;
13    }
14 };
15
16 class Queue {
17     Node* front;
18     Node* rear;
19
20 public:
21     Queue() {
22         front = rear = nullptr;
23     }
24
25     void enqueue(int value) {
26         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

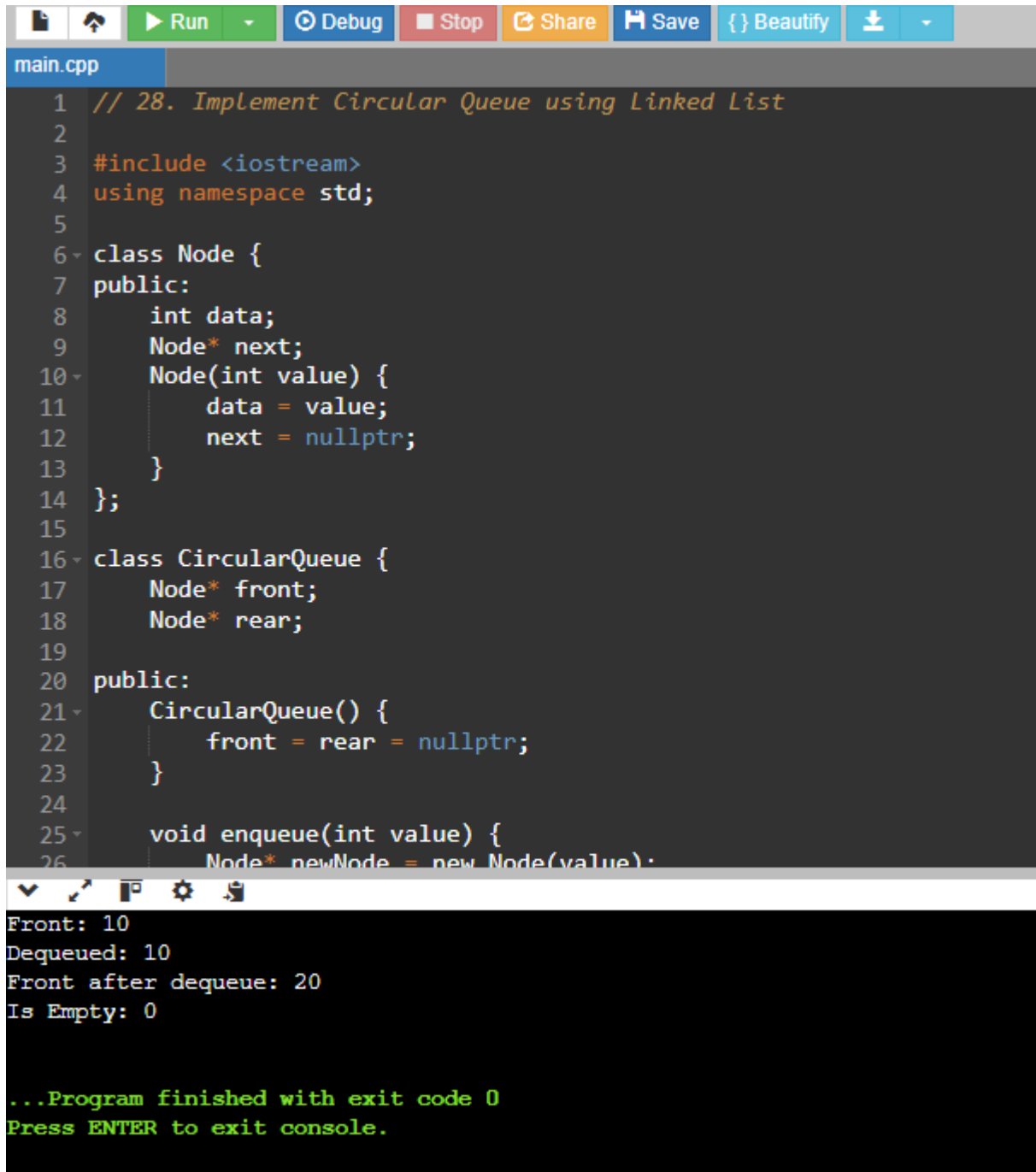


```
main.cpp
1 // 27. Implement Deque using Doubly Linked List
2
3 #include <iostream>
4 using namespace std;
5
6 class Node {
7 public:
8     int data;
9     Node* prev;
10    Node* next;
11    Node(int value) {
12        data = value;
13        prev = next = nullptr;
14    }
15 };
16
17 class Deque {
18     Node* front;
19     Node* rear;
20
21 public:
22     Deque() {
23         front = rear = nullptr;
24     }
25
26     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



```
main.cpp
1 // 28. Implement Circular Queue using Linked List
2
3 #include <iostream>
4 using namespace std;
5
6 class Node {
7 public:
8     int data;
9     Node* next;
10    Node(int value) {
11        data = value;
12        next = nullptr;
13    }
14 };
15
16 class CircularQueue {
17     Node* front;
18     Node* rear;
19
20 public:
21     CircularQueue() {
22         front = rear = nullptr;
23     }
24
25     void enqueue(int value) {
26         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

main.cpp

```
1 // 29. Implement Min Stack using Linked List
2
3 #include <iostream>
4 #include <climits>
5 using namespace std;
6
7 class Node {
8 public:
9     int data;
10    int minVal;
11    Node* next;
12    Node(int value, int minVal) {
13        data = value;
14        this->minVal = minVal;
15        next = nullptr;
16    }
17 };
18
19 class MinStack {
20     Node* top;
21
22 public:
23     MinStack() {
24         top = nullptr;
25     }
26
27     void push(int value) {
28         Node* newnode = new Node(value, min(top->minVal, value));
29         newnode->next = top;
30         top = newnode;
31     }
32
33     void pop() {
34         if (top != nullptr)
35             top = top->next;
36     }
37
38     int getTop() {
39         if (top != nullptr)
40             return top->data;
41         return -1;
42     }
43
44     int getMin() {
45         if (top != nullptr)
46             return top->minVal;
47         return -1;
48     }
49
50     bool isEmpty() {
51         return top == nullptr;
52     }
53 }
```

▼ ↗ 🏠 ⚙️ 🗑️

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

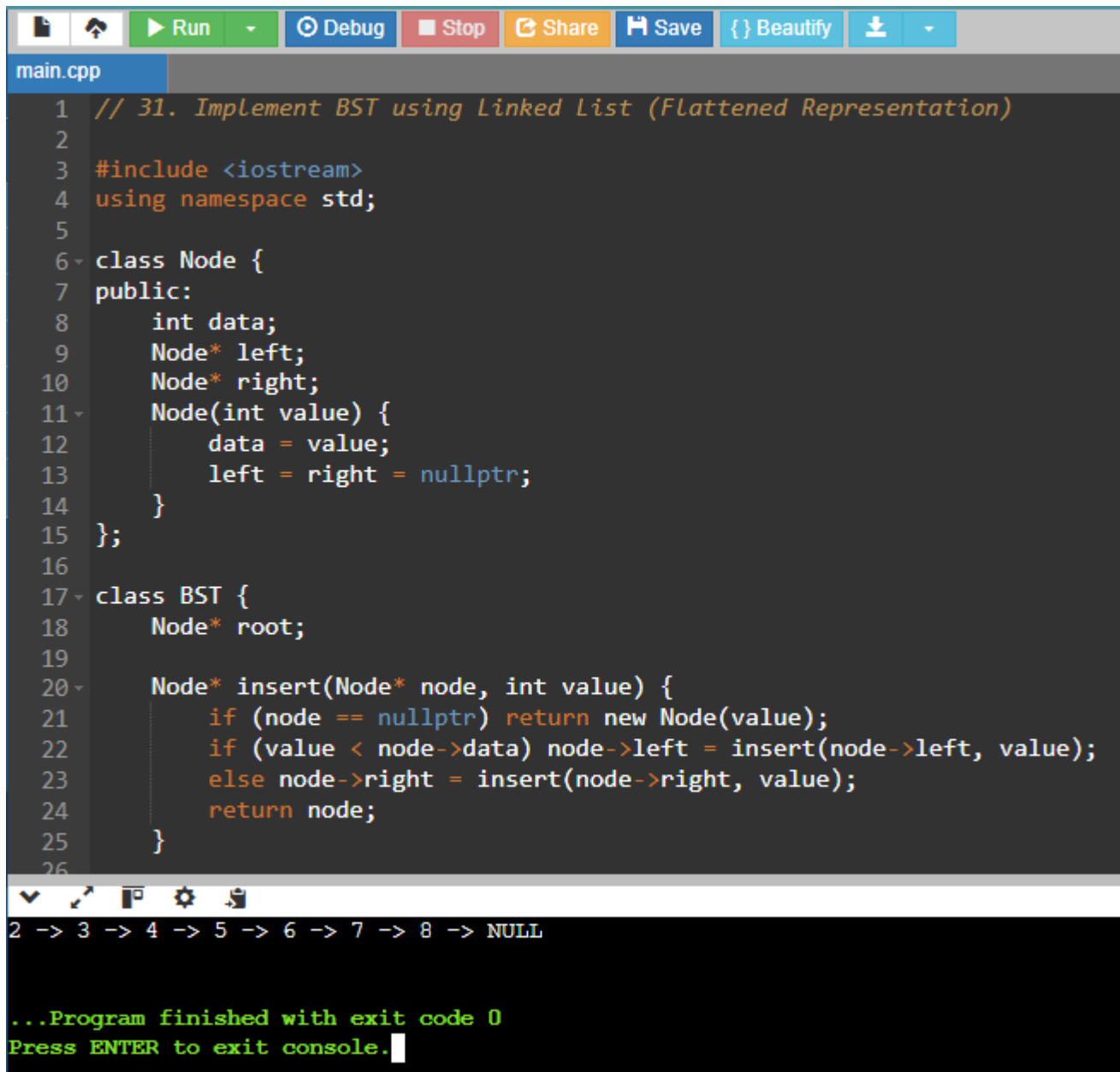
...Program finished with exit code 0
Press ENTER to exit console.

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

```
main.cpp
1 // 30. Implement Hash Table using Linked List (Chaining Method)
2
3 #include <iostream>
4 #include <list>
5 using namespace std;
6
7 class HashTable {
8     int bucketCount;
9     list<int>* table;
10
11 public:
12     HashTable(int size) {
13         bucketCount = size;
14         table = new list<int>[bucketCount];
15     }
16
17     int hashFunction(int key) {
18         return key % bucketCount;
19     }
20
21     void insert(int key) {
22         int index = hashFunction(key);
23         table[index].push_back(key);
24     }
25
26     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)



```
main.cpp
1 // 31. Implement BST using Linked List (Flattened Representation)
2
3 #include <iostream>
4 using namespace std;
5
6 class Node {
7 public:
8     int data;
9     Node* left;
10    Node* right;
11    Node(int value) {
12        data = value;
13        left = right = nullptr;
14    }
15 };
16
17 class BST {
18     Node* root;
19
20     Node* insert(Node* node, int value) {
21         if (node == nullptr) return new Node(value);
22         if (value < node->data) node->left = insert(node->left, value);
23         else node->right = insert(node->right, value);
24         return node;
25     }
26 }
27
28 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL
29
30 ...Program finished with exit code 0
31 Press ENTER to exit console.
```