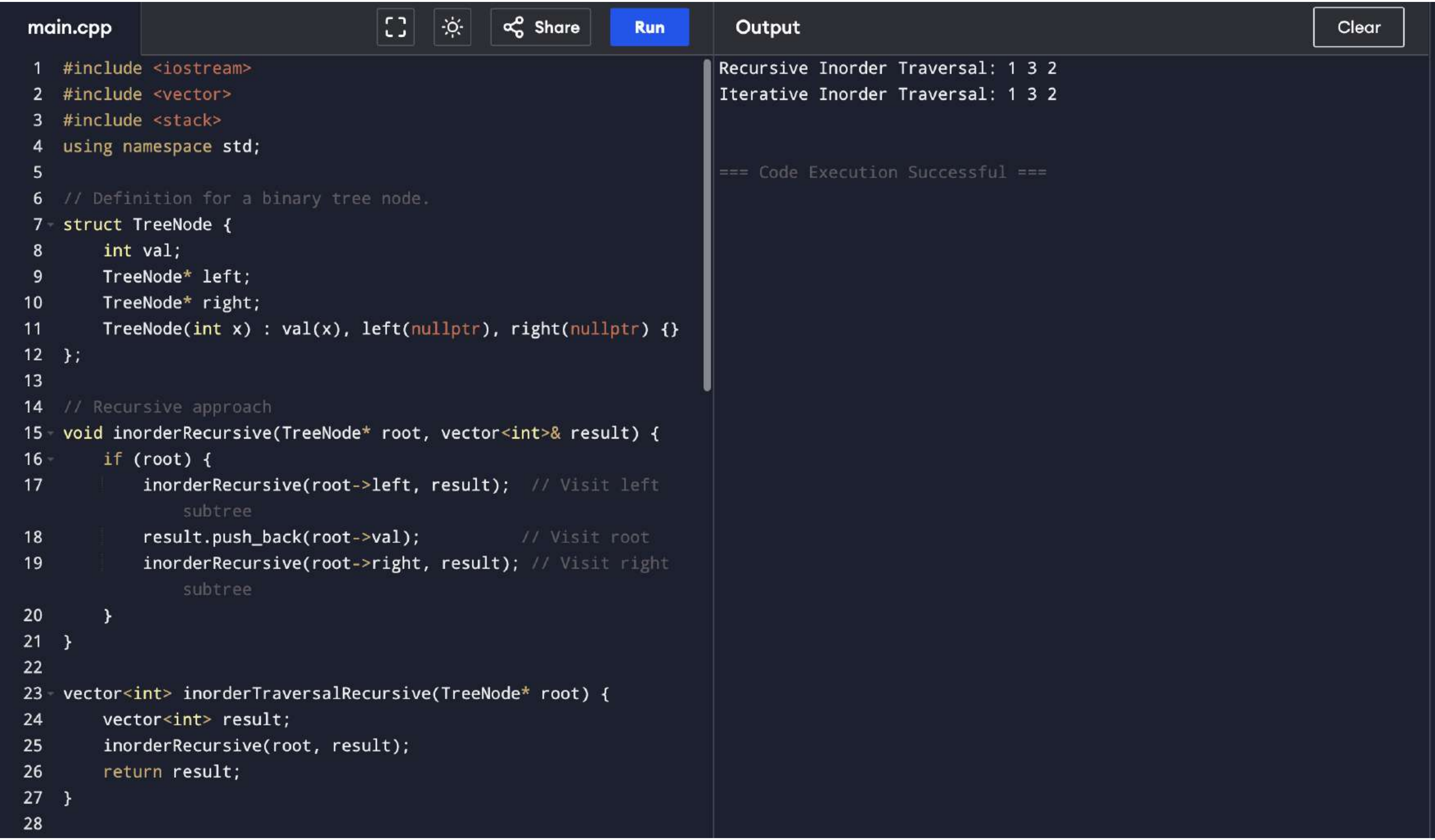
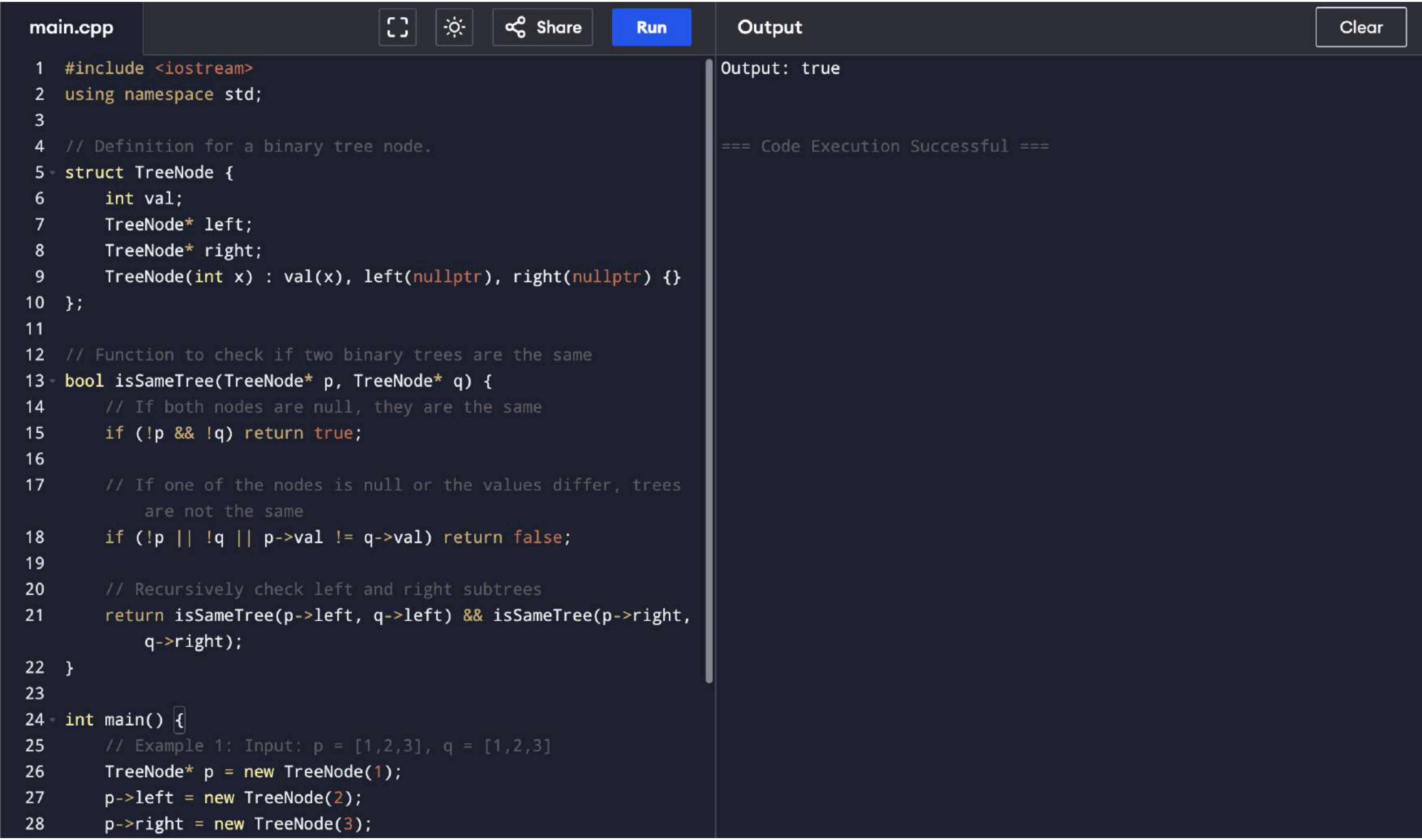


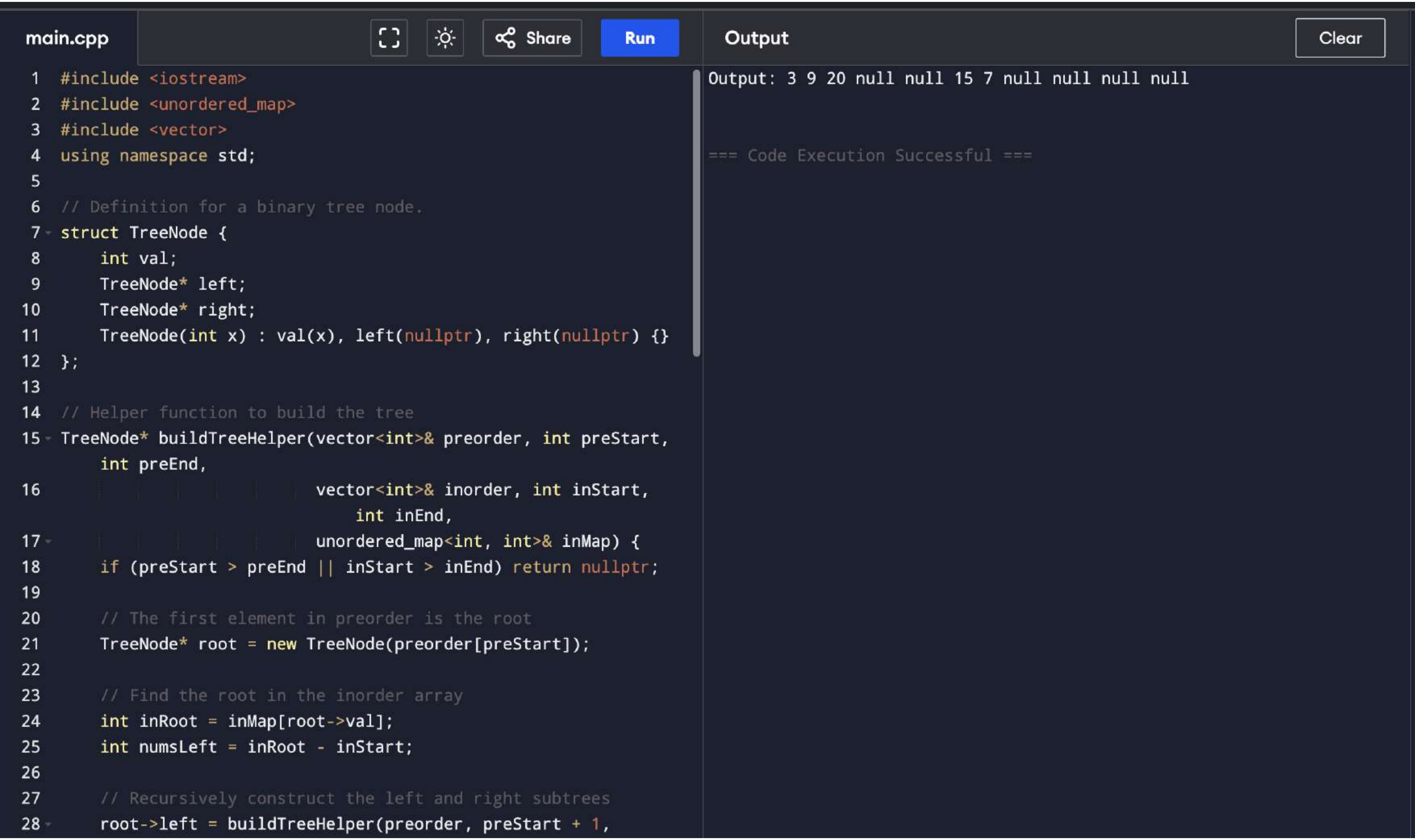
```cpp
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Recursive approach
void inorderRecursive(TreeNode* root, vector<int>& result) {
    if (root) {
        inorderRecursive(root->left, result);  // Visit left subtree
        result.push_back(root->val);          // Visit root
        inorderRecursive(root->right, result); // Visit right subtree
    }
}

vector<int> inorderTraversalRecursive(TreeNode* root) {
    vector<int> result;
    inorderRecursive(root, result);
    return result;
}

```

```
Recursive Inorder Traversal: 1 3 2
Iterative Inorder Traversal: 1 3 2


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



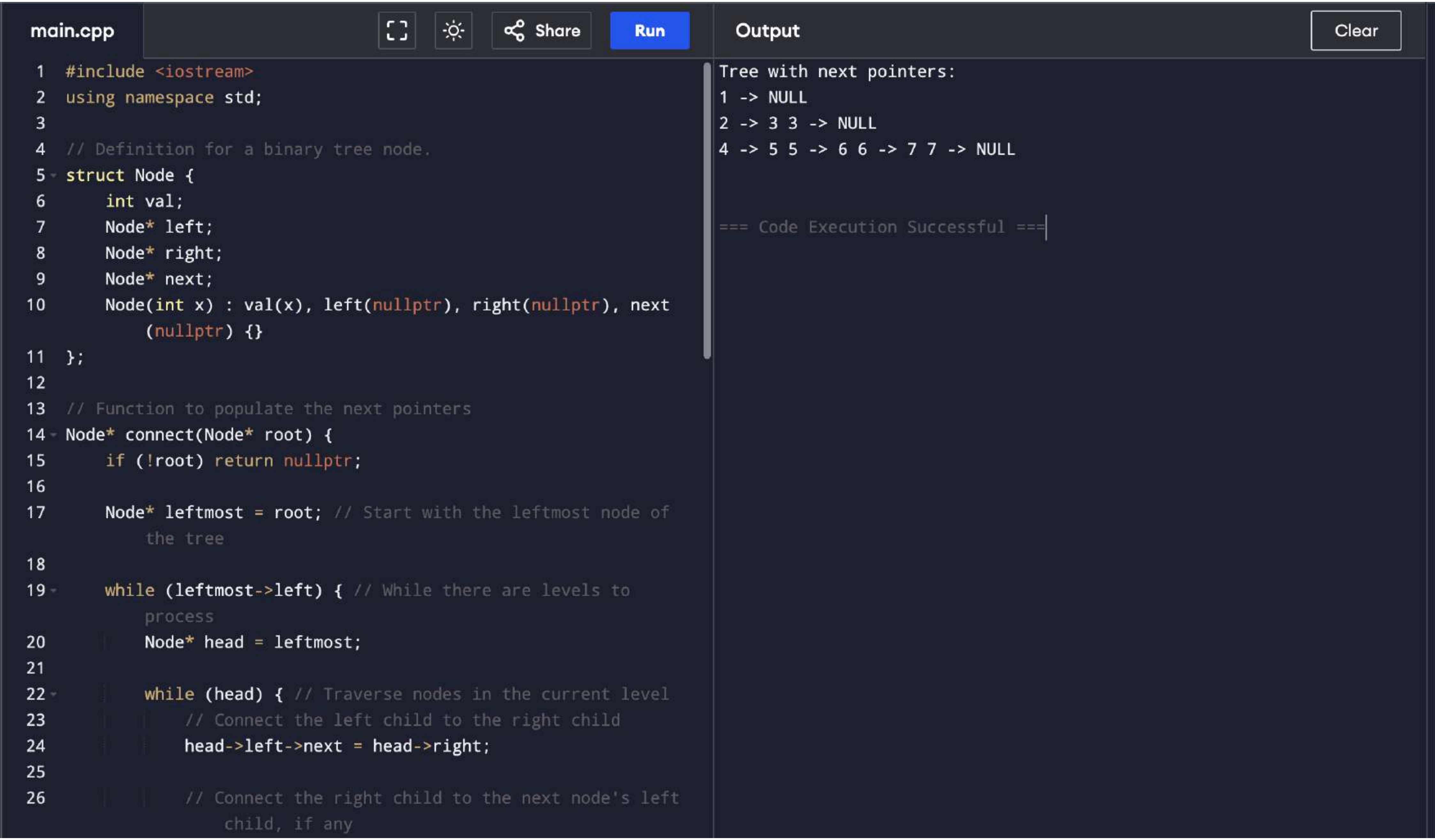
```cpp
#include <iostream>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Function to check if two binary trees are the same
bool isSameTree(TreeNode* p, TreeNode* q) {
    // If both nodes are null, they are the same
    if (!p && !q) return true;

    // If one of the nodes is null or the values differ, trees are not the same
    if (!p || !q || p->val != q->val) return false;

    // Recursively check left and right subtrees
    return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}

int main() {
    // Example 1: Input: p = [1,2,3], q = [1,2,3]
    TreeNode* p = new TreeNode(1);
    p->left = new TreeNode(2);
    p->right = new TreeNode(3);
```

```
Output: true


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



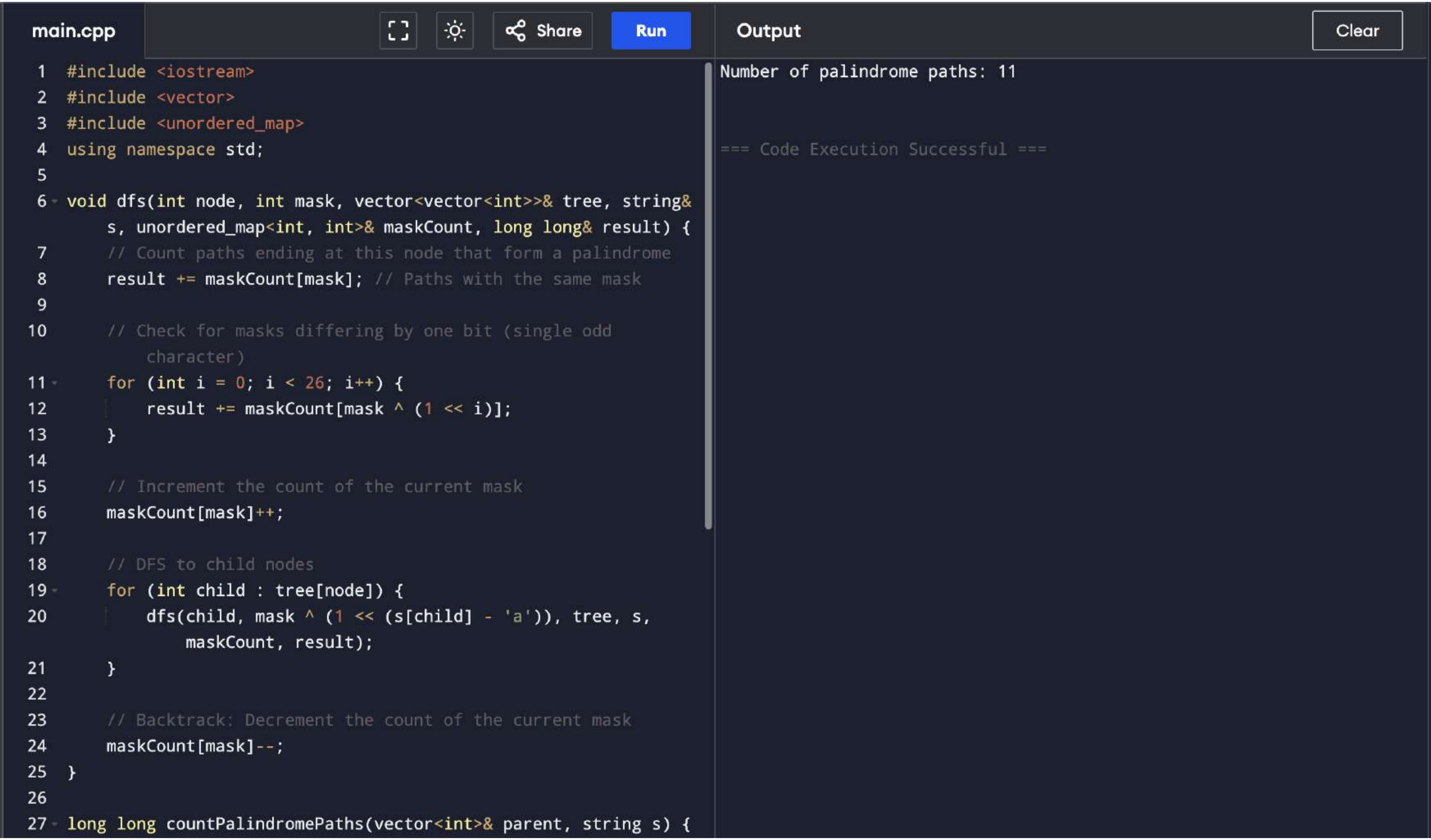
```cpp
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Helper function to build the tree
TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int preEnd,
                          vector<int>& inorder, int inStart, int inEnd,
                          unordered_map<int, int>& inMap) {
    if (preStart > preEnd || inStart > inEnd) return nullptr;

    // The first element in preorder is the root
    TreeNode* root = new TreeNode(preorder[preStart]);

    // Find the root in the inorder array
    int inRoot = inMap[root->val];
    int numsLeft = inRoot - inStart;

    // Recursively construct the left and right subtrees
    root->left = buildTreeHelper(preorder, preStart + 1,
```

```
Output: 3 9 20 null null 15 7 null null null null


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



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

// Definition for a binary tree node.
struct Node {
    int val;
    Node* left;
    Node* right;
    Node* next;
    Node(int x) : val(x), left(nullptr), right(nullptr), next(nullptr) {}
};

// Function to populate the next pointers
Node* connect(Node* root) {
    if (!root) return nullptr;

    Node* leftmost = root; // Start with the leftmost node of the tree

    while (leftmost->left) { // While there are levels to process
        Node* head = leftmost;

        while (head) { // Traverse nodes in the current level
            // Connect the left child to the right child
            head->left->next = head->right;

            // Connect the right child to the next node's left child, if any
```

```
Tree with next pointers:
1 -> NULL
2 -> 3 3 -> NULL
4 -> 5 5 -> 6 6 -> 7 7 -> NULL


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



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

void dfs(int node, int mask, vector<vector<int>>& tree, string& s, unordered_map<int, int>& maskCount, long long& result) {
    // Count paths ending at this node that form a palindrome
    result += maskCount[mask]; // Paths with the same mask

    // Check for masks differing by one bit (single odd character)
    for (int i = 0; i < 26; i++) {
        result += maskCount[mask ^ (1 << i)];
    }

    // Increment the count of the current mask
    maskCount[mask]++;

    // DFS to child nodes
    for (int child : tree[node]) {
        dfs(child, mask ^ (1 << (s[child] - 'a')), tree, s, maskCount, result);
    }

    // Backtrack: Decrement the count of the current mask
    maskCount[mask]--;
}

long long countPalindromePaths(vector<int>& parent, string s) {
```

```
Number of palindrome paths: 11


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