VERY EASY

1. Find All Paths in a Maze

You are given a maze represented as a 2D grid of integers. The maze contains cells that are either open (1) or blocked (0). Your task is to find all possible paths from the start at the top-left corner (0,0)(0, 0)(0,0) to the end at the bottom-right corner (n−1,m−1)(n-1, m-1)(n−1,m−1), moving only through open cells.

A valid path is one where:
You start at the top-left corner and end at the bottom-right corner.
You move only through cells containing 1.
You move only up, down, left, or right (no diagonal moves).
You do not visit any cell more than once in a single path.
If there is no valid path from start to end, return a message indicating that no path exists.

Example 1:
Input:
Maze:

```
1 0 1
1 1 1
0 1 1
```

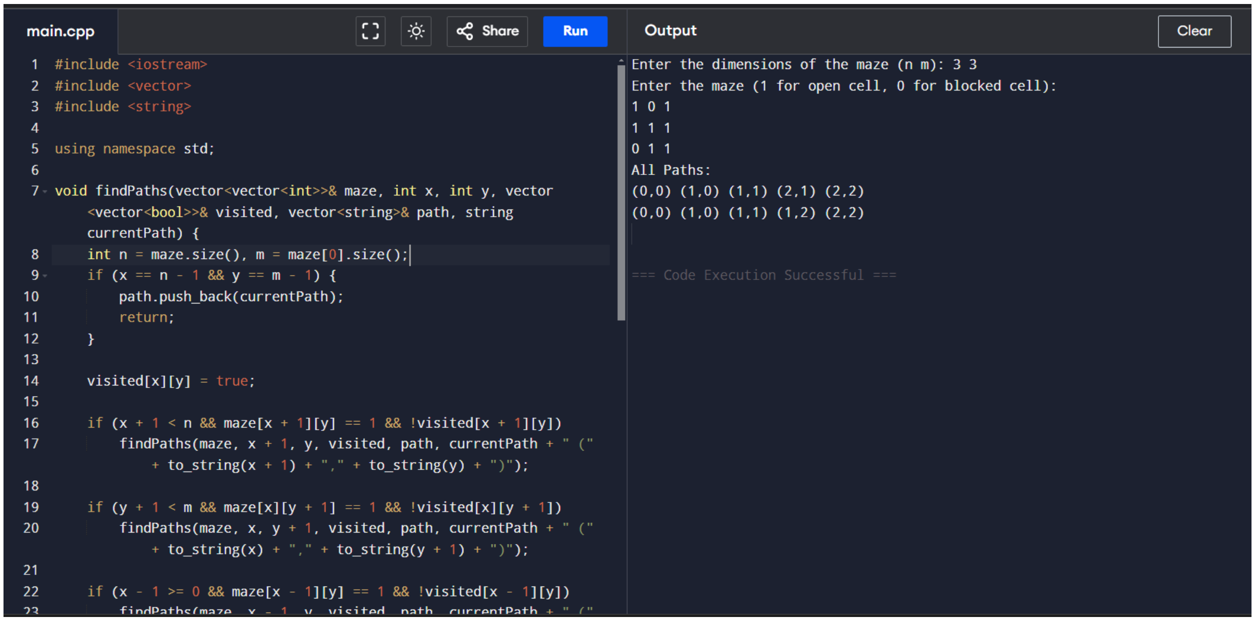
Output:
All Paths:

```
(0,0) (1,0) (1,1) (1,2) (2,2)
(0,0) (1,0) (1,1) (2,1) (2,2)
```

Code:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void findPaths(vector<vector<int>>& maze, int x, int y, vector<vector<bool>>& visited,
vector<string>& path, string currentPath) {
    int n = maze.size(), m = maze[0].size();
    if (x == n - 1 && y == m - 1) {
        path.push_back(currentPath);
        return;
    }

    visited[x][y] = true;

    if (x + 1 < n && maze[x + 1][y] == 1 && !visited[x + 1][y])
        findPaths(maze, x + 1, y, visited, path, currentPath + " (" + to_string(x + 1) + "," + to_string(y)
+ ")");

    if (y + 1 < m && maze[x][y + 1] == 1 && !visited[x][y + 1])
        findPaths(maze, x, y + 1, visited, path, currentPath + " (" + to_string(x) + "," + to_string(y + 1)
+ ")");

    if (x - 1 >= 0 && maze[x - 1][y] == 1 && !visited[x - 1][y])
        findPaths(maze, x - 1, y, visited, path, currentPath + " (" + to_string(x - 1) + "," + to_string(y) +
")");

    if (y - 1 >= 0 && maze[x][y - 1] == 1 && !visited[x][y - 1])
        findPaths(maze, x, y - 1, visited, path, currentPath + " (" + to_string(x) + "," + to_string(y - 1) +
")");

    visited[x][y] = false;
}

int main() {
    int n, m;
    cout << "Enter the dimensions of the maze (n m): ";
    cin >> n >> m;

    vector<vector<int>> maze(n, vector<int>(m));
    cout << "Enter the maze (1 for open cell, 0 for blocked cell):" << endl;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> maze[i][j];

    vector<vector<bool>> visited(n, vector<bool>(m, false));
    vector<string> path;
    string currentPath = "(0,0)";

    if (maze[0][0] == 1 && maze[n - 1][m - 1] == 1)
        findPaths(maze, 0, 0, visited, path, currentPath);

    if (path.empty()) {
        cout << "No Path Exists" << endl;
    } else {
        cout << "All Paths:" << endl;
        for (const auto& p : path)
            cout << p << endl;
    }

    return 0;
}
```

Output:



```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void findPaths(vector<vector<int>>& maze, int x, int y, vector
    <vector<bool>>& visited, vector<string>& path, string
    currentPath) {
    int n = maze.size(), m = maze[0].size();
    if (x == n - 1 && y == m - 1) {
        path.push_back(currentPath);
        return;
    }

    visited[x][y] = true;

    if (x + 1 < n && maze[x + 1][y] == 1 && !visited[x + 1][y])
        findPaths(maze, x + 1, y, visited, path, currentPath + " ("
            + to_string(x + 1) + "," + to_string(y) + ")");

    if (y + 1 < m && maze[x][y + 1] == 1 && !visited[x][y + 1])
        findPaths(maze, x, y + 1, visited, path, currentPath + " ("
            + to_string(x) + "," + to_string(y + 1) + ")");

    if (x - 1 >= 0 && maze[x - 1][y] == 1 && !visited[x - 1][y])
        findPaths(maze, x - 1, y, visited, path, currentPath + " ("
```

```
Enter the dimensions of the maze (n m): 3 3
Enter the maze (1 for open cell, 0 for blocked cell):
1 0 1
1 1 1
0 1 1
All Paths:
(0,0) (1,0) (1,1) (2,1) (2,2)
(0,0) (1,0) (1,1) (1,2) (2,2)


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

2. Generate Numbers with a Given Sum

Generate all numbers of length n whose digits sum up to a target value sum, The digits of the number will be between 0 and 9, and we will generate combinations of digits such that their sum equals the target.

Example 1:

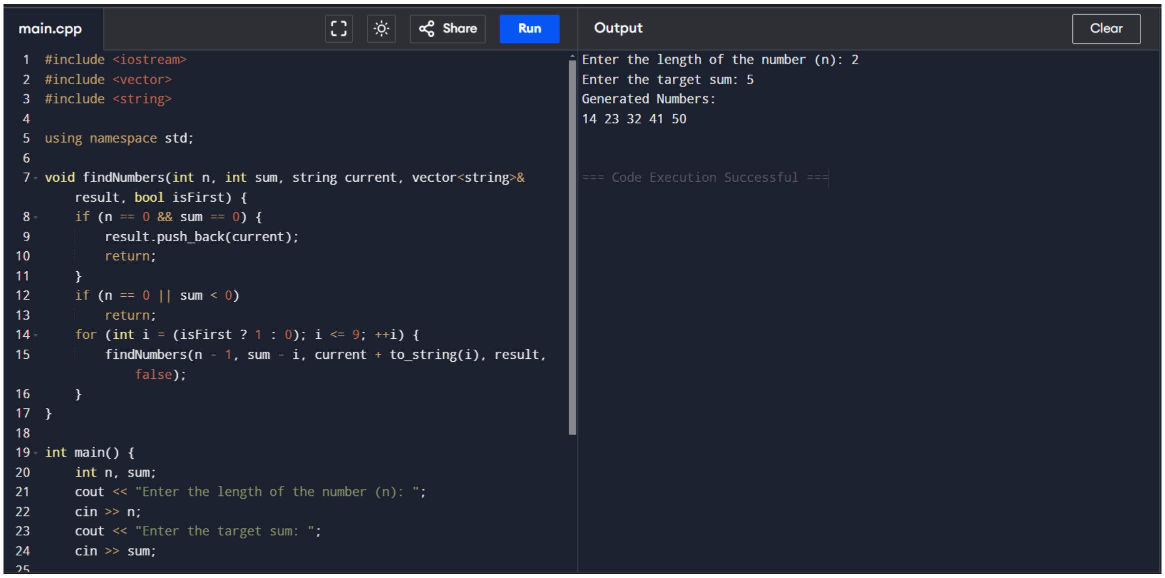
Input: n = 2 and sum = 5

Output: 14 23 32 41 50

Code:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
void findNumbers(int n, int sum, string current, vector<string>& result, bool isFirst) {
    if (n == 0 && sum == 0) {
        result.push_back(current);
        return;
    }
    if (n == 0 || sum < 0)    return;
    for (int i = (isFirst ? 1 : 0); i <= 9; ++i) {
        findNumbers(n - 1, sum - i, current + to_string(i), result, false);
    }
}
int main() {
    int n, sum;
    cout << "Enter the length of the number (n): ";
cin >> n;
cout << "Enter the target sum: ";
cin >> sum;

vector<string> result;
findNumbers(n, sum, "", result, true);
if (result.empty()) {
    cout << "No numbers possible" << endl;
} else {
    cout << "Generated Numbers:" << endl;
    for (const auto& num : result)
        cout << num << " ";
    cout << endl;
}
return 0;}
```

OUTPUT:



```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void findNumbers(int n, int sum, string current, vector<string>&
    result, bool isFirst) {
    if (n == 0 && sum == 0) {
        result.push_back(current);
        return;
    }
    if (n == 0 || sum < 0)
        return;
    for (int i = (isFirst ? 1 : 0); i <= 9; ++i) {
        findNumbers(n - 1, sum - i, current + to_string(i), result,
            false);
    }
}

int main() {
    int n, sum;
    cout << "Enter the length of the number (n): ";
    cin >> n;
    cout << "Enter the target sum: ";
    cin >> sum;
```

```
Enter the length of the number (n): 2
Enter the target sum: 5
Generated Numbers:
14 23 32 41 50


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

3. Generate Binary Strings

A binary string consists of only 0s and 1s, and our goal is to generate all possible combinations of these digits for a string of length n.

Example 1:

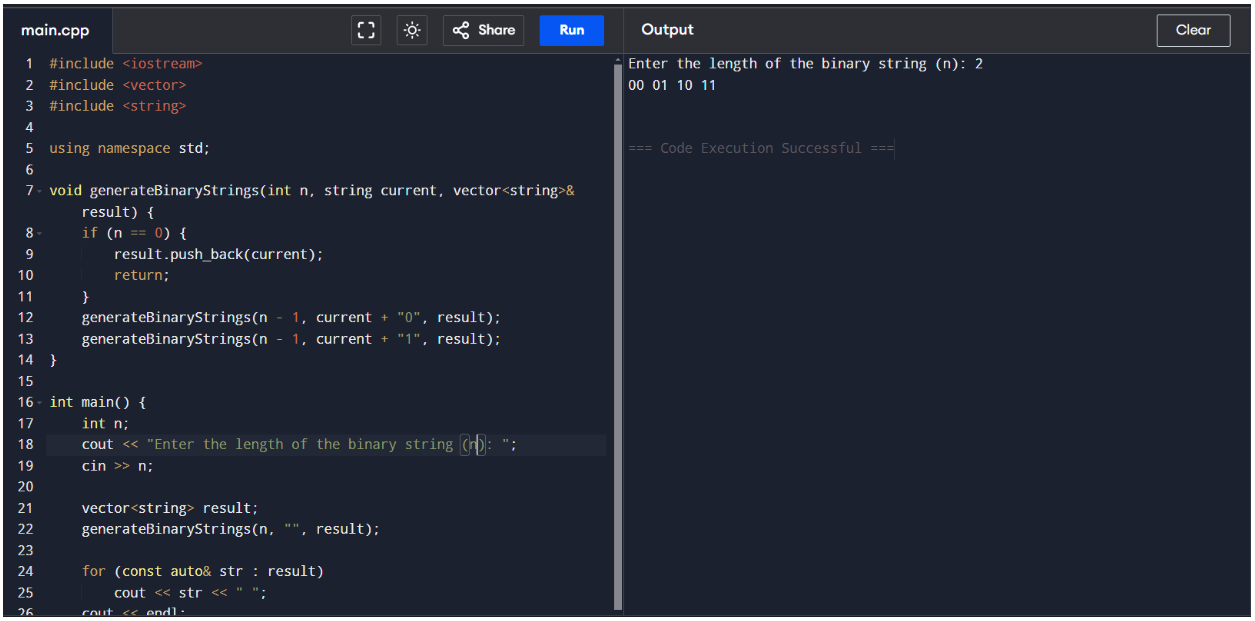
Input: n = 2

Output: 00 01 10 11

Code:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void generateBinaryStrings(int n, string current, vector<string>& result) {
    if (n == 0) {
        result.push_back(current);
        return;
    }
    generateBinaryStrings(n - 1, current + "0", result);
    generateBinaryStrings(n - 1, current + "1", result);
}

int main() {
    int n;
    cout << "Enter the length of the binary string (n): ";
    cin >> n;
    vector<string> result;
    generateBinaryStrings(n, "", result);

    for (const auto& str : result)
        cout << str << " ";
    cout << endl;

    return 0;
}
```

OUTPUT:



```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void generateBinaryStrings(int n, string current, vector<string>&
    result) {
    if (n == 0) {
        result.push_back(current);
        return;
    }
    generateBinaryStrings(n - 1, current + "0", result);
    generateBinaryStrings(n - 1, current + "1", result);
}

int main() {
    int n;
    cout << "Enter the length of the binary string (n): ";
    cin >> n;

    vector<string> result;
    generateBinaryStrings(n, "", result);

    for (const auto& str : result)
        cout << str << " ";
    cout << endl;
```

```
Enter the length of the binary string (n): 2
00 01 10 11


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

Easy:

1. Binary Tree Paths

Given the root of a binary tree, return all root-to-leaf paths in any order.

A leaf is a node with no children.

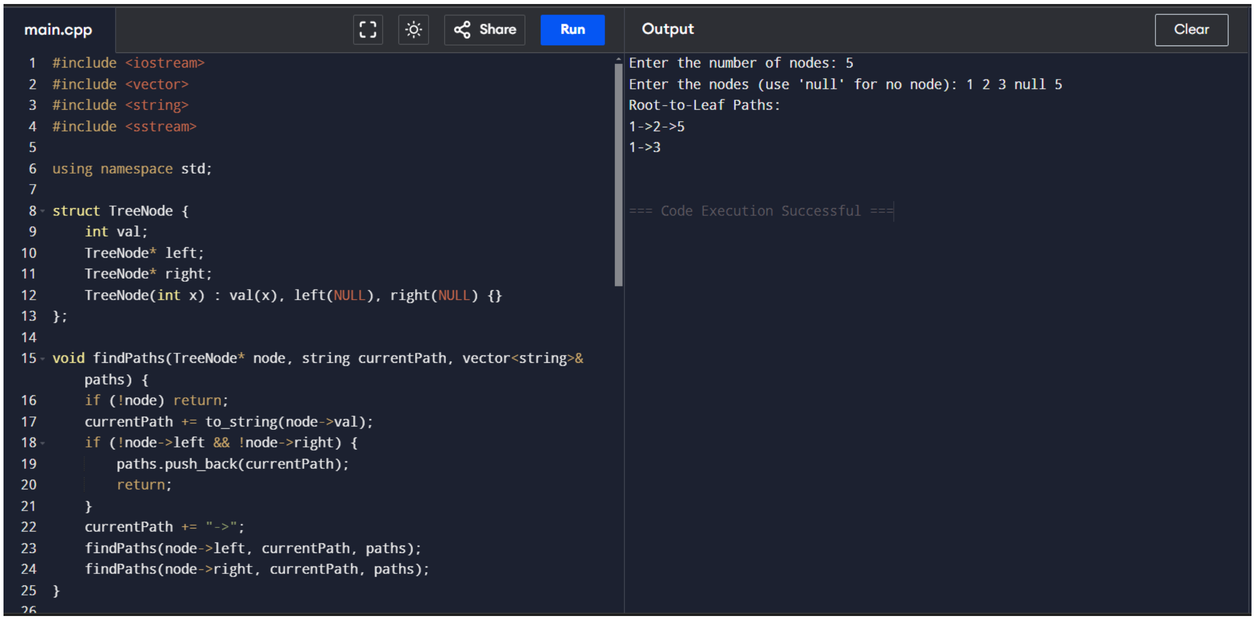
Example 1:

Input: root = [1,2,3,null,5]Output: ["1->2->5","1->3"]

CODE:

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>

using namespace std;

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

void findPaths(TreeNode* node, string currentPath, vector<string>& paths) {
    if (!node) return;
    currentPath += to_string(node->val);
    if (!node->left && !node->right) {
        paths.push_back(currentPath);
        return;
    }
    currentPath += "->";
    findPaths(node->left, currentPath, paths);
    findPaths(node->right, currentPath, paths);
}
vector<string> binaryTreePaths(TreeNode* root) {
    vector<string> paths;
    findPaths(root, "", paths);
    return paths;
}

TreeNode* buildTree(const vector<string>& nodes) {
    if (nodes.empty() || nodes[0] == "null") return nullptr;
    TreeNode* root = new TreeNode(stoi(nodes[0]));
    vector<TreeNode*> level{root};
    int i = 1;
    while (i < nodes.size()) {
        vector<TreeNode*> nextLevel;
        for (auto node : level) {
            if (i < nodes.size() && nodes[i] != "null") {
                node->left = new TreeNode(stoi(nodes[i]));
                nextLevel.push_back(node->left);
            }
            i++;
            if (i < nodes.size() && nodes[i] != "null") {
                node->right = new TreeNode(stoi(nodes[i]));
                nextLevel.push_back(node->right);
            }
            i++;
        }
        level = nextLevel;
    }
    return root;
}

int main() {
    int n;
    cout << "Enter the number of nodes: ";
    cin >> n;
    vector<string> nodes(n);
    cout << "Enter the nodes (use 'null' for no node): ";
    for (int i = 0; i < n; ++i) cin >> nodes[i];

    TreeNode* root = buildTree(nodes);
    vector<string> paths = binaryTreePaths(root);

    cout << "Root-to-Leaf Paths:" << endl;
    for (const string& path : paths) {
        cout << path << endl;
    }
    return 0;
}
```

OUTPUT:



```cpp
#include <iostream>
#include <vector>
#include <string>
#include <sstream>

using namespace std;

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

void findPaths(TreeNode* node, string currentPath, vector<string>& paths) {
    if (!node) return;
    currentPath += to_string(node->val);
    if (!node->left && !node->right) {
        paths.push_back(currentPath);
        return;
    }
    currentPath += "->";
    findPaths(node->left, currentPath, paths);
    findPaths(node->right, currentPath, paths);
}
```

```
Enter the number of nodes: 5
Enter the nodes (use 'null' for no node): 1 2 3 null 5
Root-to-Leaf Paths:
1->2->5
1->3


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

2. Permutations

Given an array nums of distinct integers, return all the possible permutations. You can return the answer in any order.

Example 1:

Input: nums = [1,2,3]

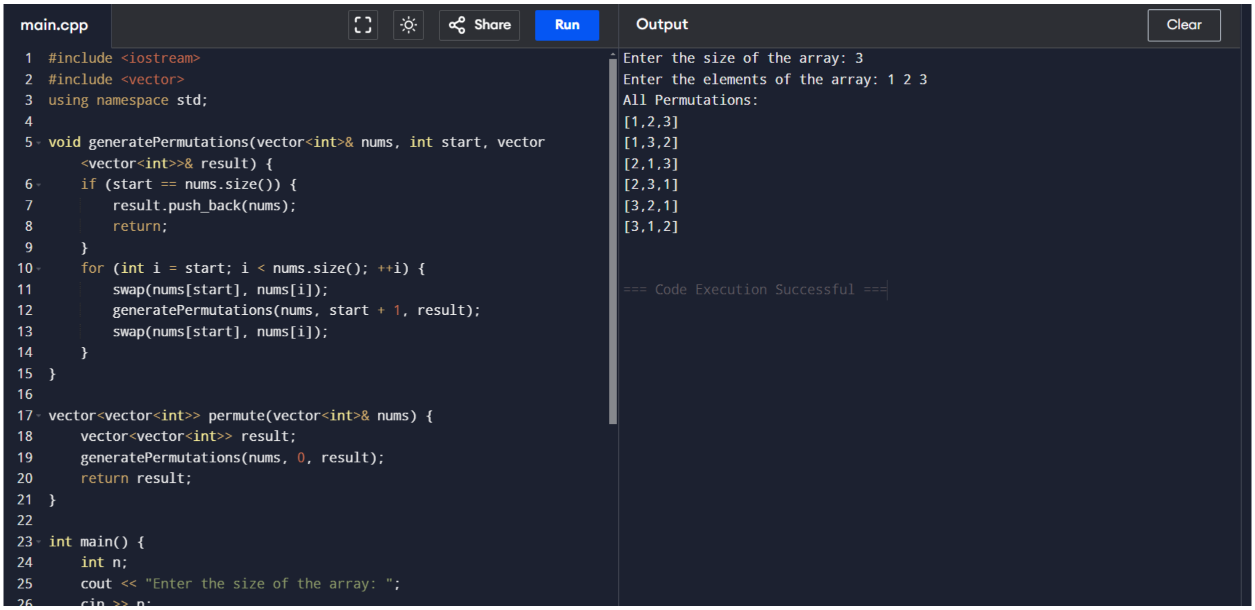
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

CODE:

```cpp
#include <iostream>

#include <vector>

using namespace std;


void generatePermutations(vector<int>& nums, int start, vector<vector<int>>& result) {

   if (start == nums.size()) {

       result.push_back(nums);
```

```
        return;
    }
    for (int i = start; i < nums.size(); ++i) {
        swap(nums[start], nums[i]);
        generatePermutations(nums, start + 1, result);
        swap(nums[start], nums[i]);
    }
}

vector<vector<int>> permute(vector<int>& nums) {
    vector<vector<int>> result;
    generatePermutations(nums, 0, result);
    return result;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) cin >> nums[i];

    vector<vector<int>> permutations = permute(nums);

    cout << "All Permutations:" << endl;
    for (const auto& perm : permutations) {
        cout << "[";
        for (int i = 0; i < perm.size(); ++i) {
            cout << perm[i];
            if (i < perm.size() - 1) cout << ",";
        }
        cout << "]" << endl;
    }

    return 0;
}
```

OUTPUT:



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

void generatePermutations(vector<int>& nums, int start, vector
    <vector<int>>& result) {
    if (start == nums.size()) {
        result.push_back(nums);
        return;
    }
    for (int i = start; i < nums.size(); ++i) {
        swap(nums[start], nums[i]);
        generatePermutations(nums, start + 1, result);
        swap(nums[start], nums[i]);
    }
}

vector<vector<int>> permute(vector<int>& nums) {
    vector<vector<int>> result;
    generatePermutations(nums, 0, result);
    return result;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
```

```
Enter the size of the array: 3
Enter the elements of the array: 1 2 3
All Permutations:
[1,2,3]
[1,3,2]
[2,1,3]
[2,3,1]
[3,2,1]
[3,1,2]


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

3. Subsets

Given an integer array nums of unique elements, return all possible subsets (the power set).

The solution set must not contain duplicate subsets. Return the solution in any order.

Example 1:

Input: nums = [1,2,3]

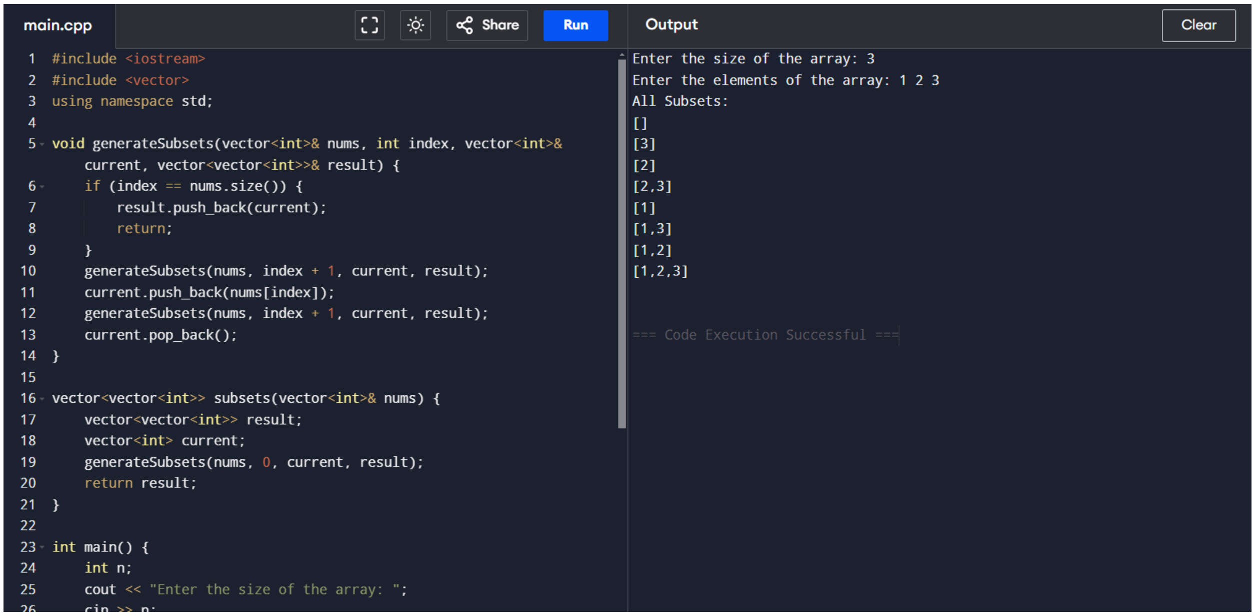
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

CODE:

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

void generateSubsets(vector<int>& nums, int index, vector<int>& current, vector<vector<int>>& result) {
   if (index == nums.size()) {
      result.push_back(current);
      return;
   }
   generateSubsets(nums, index + 1, current, result);
   current.push_back(nums[index]);
   generateSubsets(nums, index + 1, current, result);
   current.pop_back();
}

vector<vector<int>> subsets(vector<int>& nums) {
   vector<vector<int>> result;
   vector<int> current;
```

```
    generateSubsets(nums, 0, current, result);
    return result;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) cin >> nums[i];

    vector<vector<int>> result = subsets(nums);

    cout << "All Subsets:" << endl;
    for (const auto& subset : result) {
        cout << "[";
        for (int i = 0; i < subset.size(); ++i) {
            cout << subset[i];
            if (i < subset.size() - 1) cout << ",";
        }
        cout << "]" << endl;
    }
    return 0;
}
```

OUTPUT:



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

void generateSubsets(vector<int>& nums, int index, vector<int>&
    current, vector<vector<int>>& result) {
    if (index == nums.size()) {
        result.push_back(current);
        return;
    }
    generateSubsets(nums, index + 1, current, result);
    current.push_back(nums[index]);
    generateSubsets(nums, index + 1, current, result);
    current.pop_back();
}

vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> result;
    vector<int> current;
    generateSubsets(nums, 0, current, result);
    return result;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
```

```
Enter the size of the array: 3
Enter the elements of the array: 1 2 3
All Subsets:
[]
[3]
[2]
[2,3]
[1]
[1,3]
[1,2]
[1,2,3]


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

Medium:

1. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

Input: digits = "23"

Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

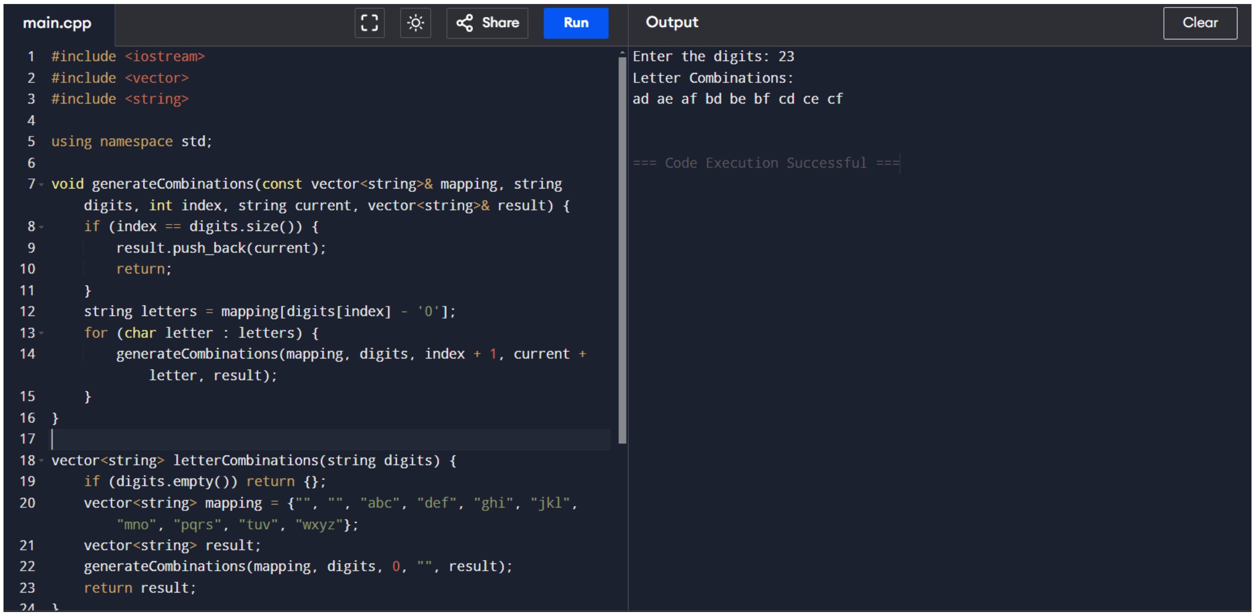
CODE:

#include <iostream>

#include <vector>

```
#include <string>

using namespace std;

void generateCombinations(const vector<string>& mapping, string digits, int index, string current, vector<string>& result) {
    if (index == digits.size()) {
        result.push_back(current);
        return;
    }
    string letters = mapping[digits[index] - '0'];
    for (char letter : letters) {
        generateCombinations(mapping, digits, index + 1, current + letter, result);
    }
}

vector<string> letterCombinations(string digits) {
    if (digits.empty()) return {};
    vector<string> mapping = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
    vector<string> result;
    generateCombinations(mapping, digits, 0, "", result);
    return result;
}

int main() {
    string digits;
    cout << "Enter the digits: ";
    cin >> digits;


    vector<string> combinations = letterCombinations(digits);


    cout << "Letter Combinations:" << endl;
    for (const auto& combination : combinations) {
        cout << combination << " ";
    }
    cout << endl;


    return 0;
}
```

OUTPUT:



```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void generateCombinations(const vector<string>& mapping, string digits, int index, string current, vector<string>& result) {
    if (index == digits.size()) {
        result.push_back(current);
        return;
    }
    string letters = mapping[digits[index] - '0'];
    for (char letter : letters) {
        generateCombinations(mapping, digits, index + 1, current + letter, result);
    }
}

vector<string> letterCombinations(string digits) {
    if (digits.empty()) return {};
    vector<string> mapping = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
    vector<string> result;
    generateCombinations(mapping, digits, 0, "", result);
    return result;
}
```

```
Enter the digits: 23
Letter Combinations:
ad ae af bd be bf cd ce cf


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

2. Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

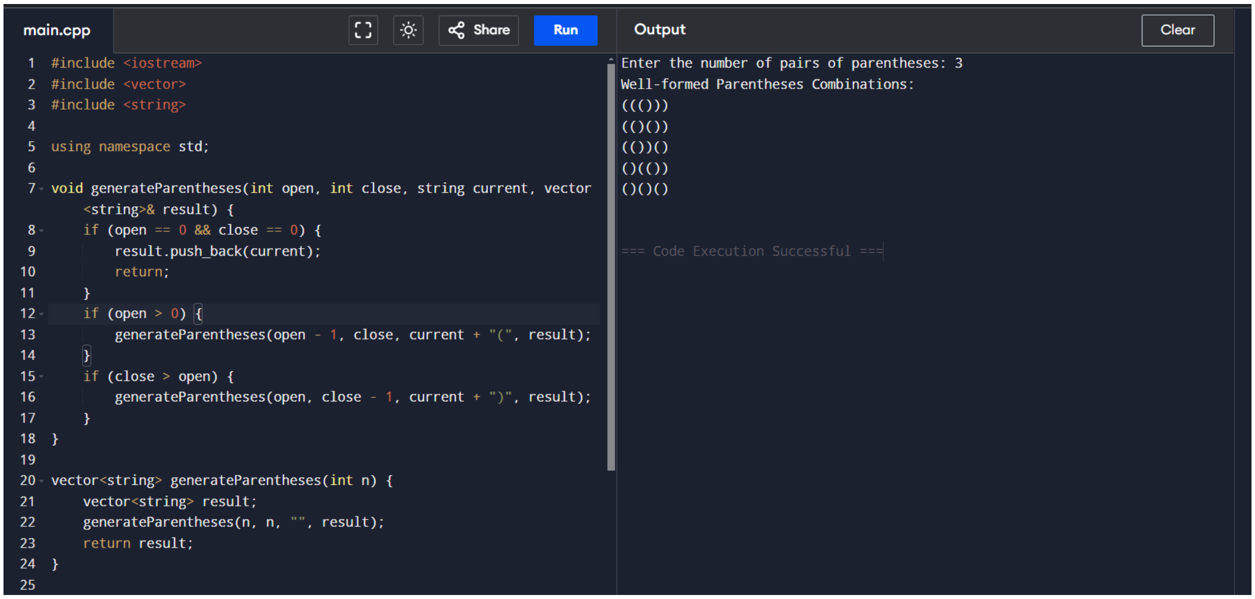
Example 1:

Input: n = 3Output: ["((()))","(()())","(())()","()(())","()()()"]

CODE:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void generateParentheses(int open, int close, string current, vector<string>& result) {
    if (open == 0 && close == 0) {
        result.push_back(current);
        return;
    }
    if (open > 0) {
        generateParentheses(open - 1, close, current + "(", result);
    }
    if (close > open) {
        generateParentheses(open, close - 1, current + ")", result);
    }
}

vector<string> generateParentheses(int n) {
    vector<string> result;
    generateParentheses(n, n, "", result);
    return result;
}


int main() {
    int n;
    cout << "Enter the number of pairs of parentheses: ";
    cin >> n;

    vector<string> combinations = generateParentheses(n);

    cout << "Well-formed Parentheses Combinations:" << endl;
    for (const auto& combination : combinations) {
        cout << combination << endl;
    }
    return 0;
}
```

OUTPUT:



```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void generateParentheses(int open, int close, string current, vector
    <string>& result) {
    if (open == 0 && close == 0) {
        result.push_back(current);
        return;
    }
    if (open > 0) {
        generateParentheses(open - 1, close, current + "(", result);
    }
    if (close > open) {
        generateParentheses(open, close - 1, current + ")", result);
    }
}

vector<string> generateParentheses(int n) {
    vector<string> result;
    generateParentheses(n, n, "", result);
    return result;
}

```

```
Enter the number of pairs of parentheses: 3
Well-formed Parentheses Combinations:
((()))
(()())
(())()
()(())
()()()


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

3. Combination Sum

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

Input: candidates = [2,3,6,7], target = 7

Output: [[2,2,3],[7]]Explanation:

2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.

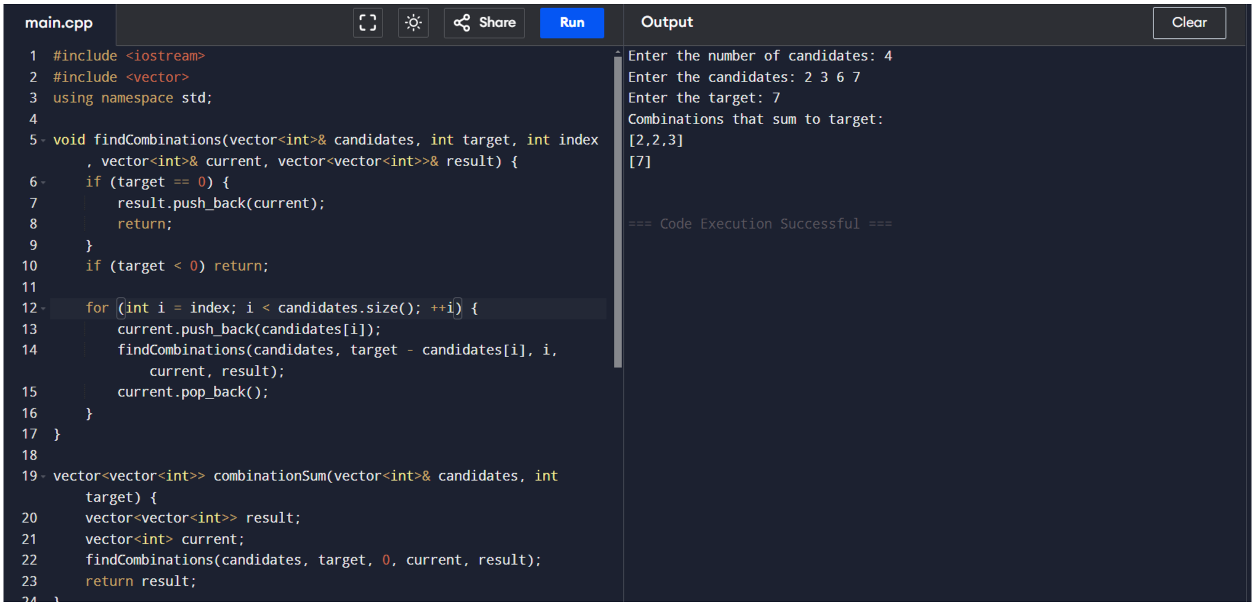
7 is a candidate, and 7 = 7.

These are the only two combinations.

CODE:

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

void findCombinations(vector<int>& candidates, int target, int index, vector<int>& current,
vector<vector<int>>& result) {
  if (target == 0) {
    result.push_back(current);
    return;
  }
  if (target < 0) return;

  for (int i = index; i < candidates.size(); ++i) {
        current.push_back(candidates[i]);

        findCombinations(candidates, target - candidates[i], i, current, result);

        current.pop_back();

    }

}


vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

    vector<vector<int>> result;

    vector<int> current;

    findCombinations(candidates, target, 0, current, result);

    return result;

}


int main() {

    int n, target;

    cout << "Enter the number of candidates: ";

    cin >> n;

    vector<int> candidates(n);

    cout << "Enter the candidates: ";

    for (int i = 0; i < n; ++i) cin >> candidates[i];

    cout << "Enter the target: ";

    cin >> target;


    vector<vector<int>> combinations = combinationSum(candidates, target);


    cout << "Combinations that sum to target:" << endl;

    for (const auto& combination : combinations) {
        cout << "[";
        for (int i = 0; i < combination.size(); ++i) {
            cout << combination[i];
            if (i < combination.size() - 1) cout << ",";
        }
        cout << "]" << endl;
    }

    return 0;
}
```

OUTPUT:



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

void findCombinations(vector<int>& candidates, int target, int index
    , vector<int>& current, vector<vector<int>>& result) {
    if (target == 0) {
        result.push_back(current);
        return;
    }
    if (target < 0) return;

    for (int i = index; i < candidates.size(); ++i) {
        current.push_back(candidates[i]);
        findCombinations(candidates, target - candidates[i], i,
            current, result);
        current.pop_back();
    }
}

vector<vector<int>> combinationSum(vector<int>& candidates, int
    target) {
    vector<vector<int>> result;
    vector<int> current;
    findCombinations(candidates, target, 0, current, result);
    return result;
}
```

```
Enter the number of candidates: 4
Enter the candidates: 2 3 6 7
Enter the target: 7
Combinations that sum to target:
[2,2,3]
[7]


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

HARD

1. N-Queens I

The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return the number of distinct solutions to the n-queens puzzle.

Example 1:

Input: n = 4

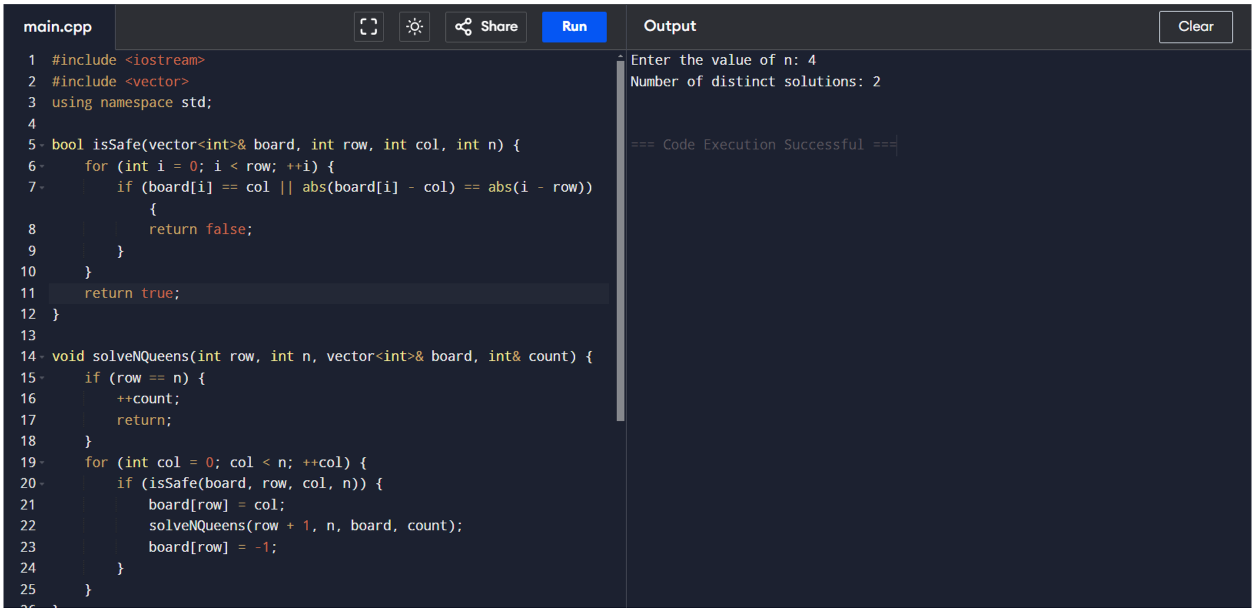
Output: 2

Explanation: There are two distinct solutions to the 4-queens puzzle as shown.

CODE:

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

bool isSafe(vector<int>& board, int row, int col, int n) {
    for (int i = 0; i < row; ++i) {
        if (board[i] == col || abs(board[i] - col) == abs(i - row)) {
            return false;
        }
    }
    return true;
}

void solveNQueens(int row, int n, vector<int>& board, int& count) {
    if (row == n) {
        ++count;
        return;
    }
    for (int col = 0; col < n; ++col) {
        if (isSafe(board, row, col, n)) {
            board[row] = col;
            solveNQueens(row + 1, n, board, count);
            board[row] = -1;
        }
    }
}

int totalNQueens(int n) {
    vector<int> board(n, -1);
    int count = 0;
    solveNQueens(0, n, board, count);
    return count;
}

int main() {
    int n;
    cout << "Enter the value of n: ";
    cin >> n;

    int result = totalNQueens(n);
    cout << "Number of distinct solutions: " << result << endl;
    return 0;
}
```

OUTPUT:



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

bool isSafe(vector<int>& board, int row, int col, int n) {
    for (int i = 0; i < row; ++i) {
        if (board[i] == col || abs(board[i] - col) == abs(i - row))
            {
            return false;
        }
    }
    return true;
}

void solveNQueens(int row, int n, vector<int>& board, int& count) {
    if (row == n) {
        ++count;
        return;
    }
    for (int col = 0; col < n; ++col) {
        if (isSafe(board, row, col, n)) {
            board[row] = col;
            solveNQueens(row + 1, n, board, count);
            board[row] = -1;
        }
    }
```

```
Enter the value of n: 4
Number of distinct solutions: 2


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

2. Restore IP Addresses

A valid IP address consists of exactly four integers separated by single dots. Each integer is between 0 and 255 (inclusive) and cannot have leading zeros.

For example, "0.1.2.201" and "192.168.1.1" are valid IP addresses, but "0.011.255.245", "192.168.1.312" and "192.168@1.1" are invalid IP addresses.

Given a string s containing only digits, return all possible valid IP addresses that can be formed by inserting dots into s. You are not allowed to reorder or remove any digits in s. You may return the valid IP addresses in any order.

Example 1:

Input: s = "25525511135"

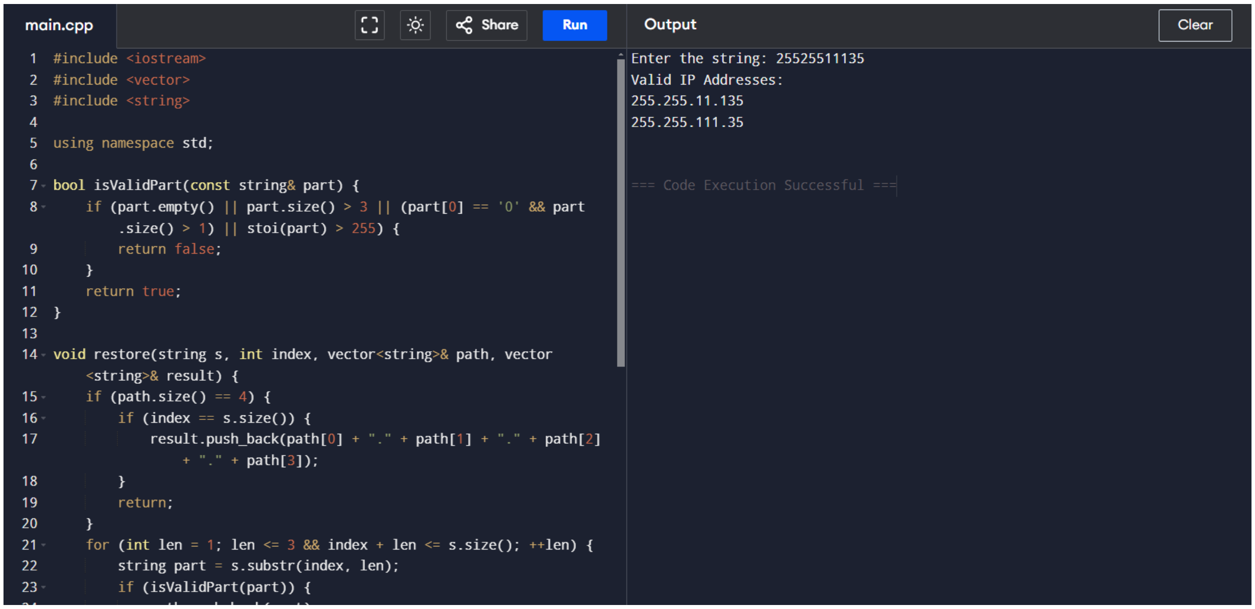
Output: ["255.255.11.135","255.255.111.35"]

CODE:

#include <iostream>

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

bool isValidPart(const string& part) {
    if (part.empty() || part.size() > 3 || (part[0] == '0' && part.size() > 1) || stoi(part) > 255) {
        return false;
    }
    return true;
}

void restore(string s, int index, vector<string>& path, vector<string>& result) {
    if (path.size() == 4) {
        if (index == s.size()) {
            result.push_back(path[0] + "." + path[1] + "." + path[2] + "." + path[3]);
        }
        return;
    }
    for (int len = 1; len <= 3 && index + len <= s.size(); ++len) {
        string part = s.substr(index, len);
        if (isValidPart(part)) {
            path.push_back(part);
            restore(s, index + len, path, result);
            path.pop_back();
        }
    }}
vector<string> restoreIpAddresses(string s) {
    vector<string> result;
    vector<string> path;
    restore(s, 0, path, result);
    return result;
}
int main() {
    string s;
    cout << "Enter the string: ";
    cin >> s;
    vector<string> ipAddresses = restoreIpAddresses(s);
    cout << "Valid IP Addresses:" << endl;
    for (const auto& ip : ipAddresses) {
        cout << ip << endl;
   } return 0;
}
```

OUTPUT:



```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

bool isValidPart(const string& part) {
    if (part.empty() || part.size() > 3 || (part[0] == '0' && part
        .size() > 1) || stoi(part) > 255) {
        return false;
    }
    return true;
}

void restore(string s, int index, vector<string>& path, vector
    <string>& result) {
    if (path.size() == 4) {
        if (index == s.size()) {
            result.push_back(path[0] + "." + path[1] + "." + path[2]
                + "." + path[3]);
        }
        return;
    }
    for (int len = 1; len <= 3 && index + len <= s.size(); ++len) {
        string part = s.substr(index, len);
        if (isValidPart(part)) {
```

```
Enter the string: 25525511135
Valid IP Addresses:
255.255.11.135
255.255.111.35

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

3. Gray Code

An n-bit gray code sequence is a sequence of 2n integers where: Every integer is in the inclusive range [0, 2n - 1],The first integer is 0,

An integer appears no more than once in the sequence,

The binary representation of every pair of adjacent integers differs by exactly one bit, and

The binary representation of the first and last integers differs by exactly one bit.

Given an integer n, return any valid n-bit gray code sequence.

Example 1:

Input: n = 2

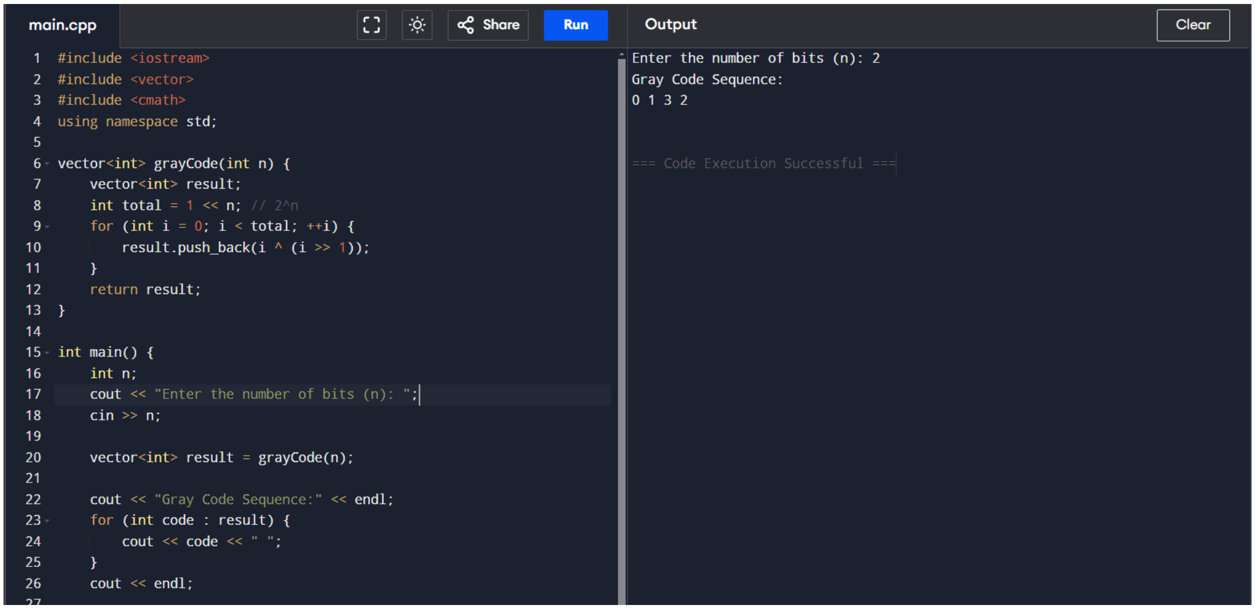
Output: [0,1,3,2]

CODE:

```
#include <iostream>

#include <vector>

#include <cmath>

using namespace std;


vector<int> grayCode(int n) {

    vector<int> result;

    int total = 1 << n; // 2^n

    for (int i = 0; i < total; ++i) {

        result.push_back(i ^ (i >> 1));

    }

    return result;

}
```

```
int main() {
    int n;
    cout << "Enter the number of bits (n): ";
    cin >> n;

    vector<int> result = grayCode(n);

    cout << "Gray Code Sequence:" << endl;
    for (int code : result) {
        cout << code << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT:



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

vector<int> grayCode(int n) {
    vector<int> result;
    int total = 1 << n; // 2^n
    for (int i = 0; i < total; ++i) {
        result.push_back(i ^ (i >> 1));
    }
    return result;
}

int main() {
    int n;
    cout << "Enter the number of bits (n): ";
    cin >> n;

    vector<int> result = grayCode(n);

    cout << "Gray Code Sequence:" << endl;
    for (int code : result) {
        cout << code << " ";
    }
    cout << endl;
```

```
Enter the number of bits (n): 2
Gray Code Sequence:
0 1 3 2


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