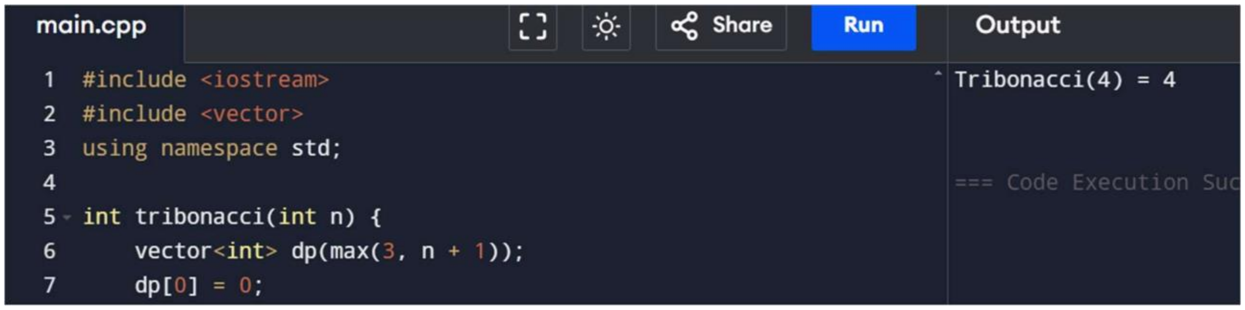
DOMAIN WINTER WINNING CAMP(Day-8)

Student Name: Shreyansh Vishmoi
Branch: BE-CSE
Semester: 6th

UID: 22BCS15373
Section/Group: 620-B
Date of Performance: 29/12/24

1) N-th Tribonacci Number

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

int tribonacci(int n) {
    vector<int> dp(max(3, n + 1));
    dp[0] = 0;
    dp[1] = dp[2] = 1;
    for (int i = 3; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];
    }
    return dp[n];
}

int main() {
    int n = 4;
    cout << "Tribonacci(" << n << ") = " << tribonacci(n) << endl;
return 0;
}
```

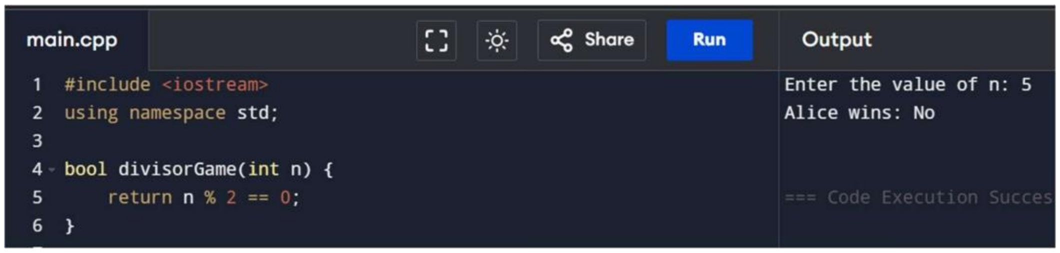


```
#include <iostream>
#include <vector>
using namespace std;

int tribonacci(int n) {
    vector<int> dp(max(3, n + 1));
    dp[0] = 0;
```

```
Tribonacci(4) = 4


=== Code Execution Suc
```

2) Divisor Game

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

bool divisorGame(int n) {
    return n % 2 == 0;
}

int main() {
    int n;
    cout << "Enter the value of n: ";
    cin >> n;
    cout << "Alice wins: " << (divisorGame(n) ? "Yes" : "No") << endl;
    return 0;
}
```



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

bool divisorGame(int n) {
    return n % 2 == 0;
}
```

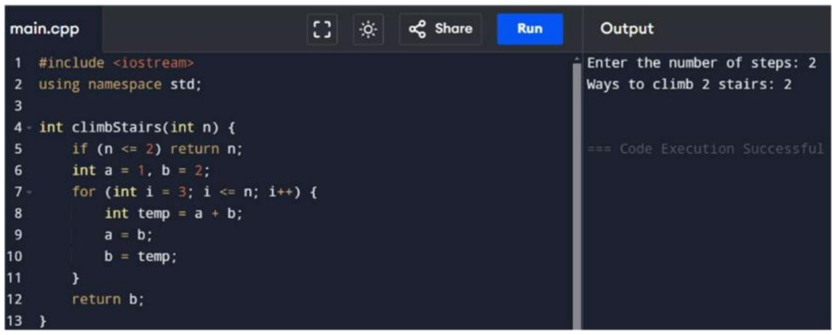
```
Enter the value of n: 5
Alice wins: No

=== Code Execution Succes
```

3) Climbing Stairs

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

int climbStairs(int n) {
    if (n <= 2) return n;
    int a = 1, b = 2;
    for (int i = 3; i <= n; i++) {
        int temp = a + b;
        a = b;
        b = temp;
    }
    return b;
}

int main() {
    int n;
    cout << "Enter the number of steps: ";
    cin >> n;
    cout << "Ways to climb " << n << " stairs: " << climbStairs(n) << endl;
    return 0;
```

}

Output :

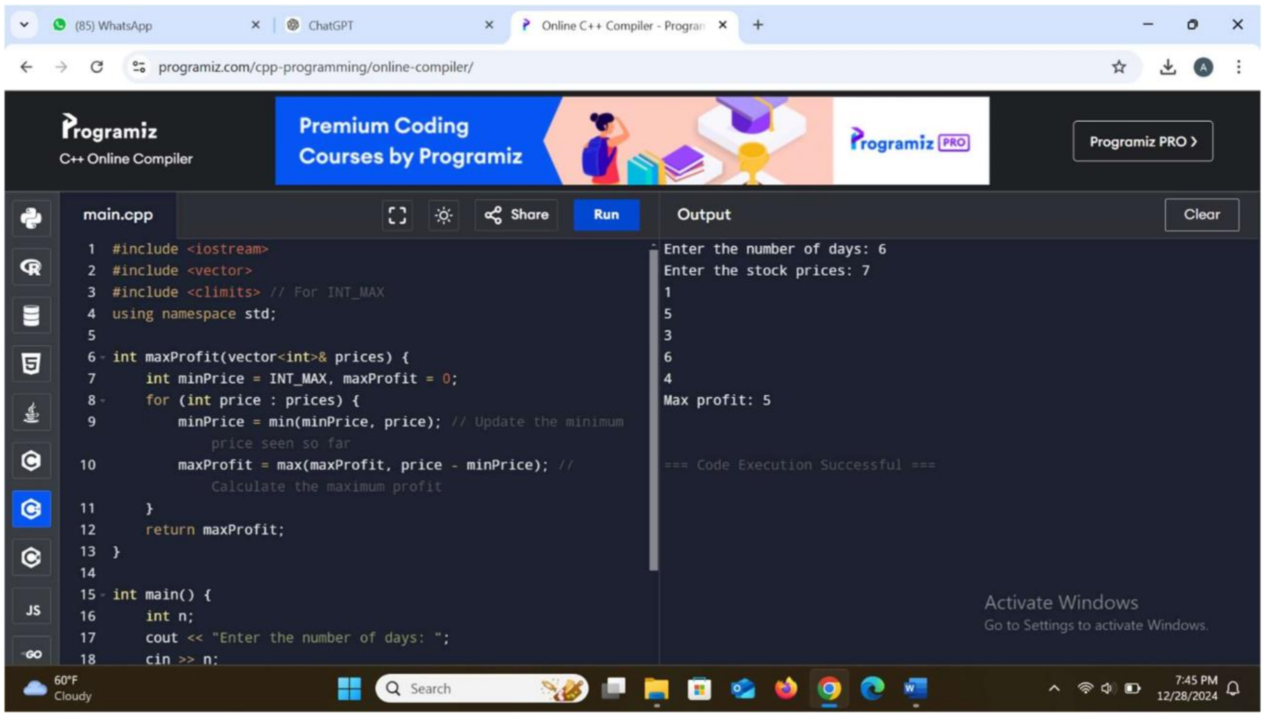


```
#include <iostream>
using namespace std;

int climbStairs(int n) {
    if (n <= 2) return n;
    int a = 1, b = 2;
    for (int i = 3; i <= n; i++) {
        int temp = a + b;
        a = b;
        b = temp;
    }
    return b;
}
```

```
Enter the number of steps: 2
Ways to climb 2 stairs: 2

=== Code Execution Successful
```

4) Best Time to Buy and Sell Stock

```
#include <iostream>
#include <vector>
#include <climits> // For INT_MAX
using namespace std;

int maxProfit(vector<int>& prices) {
    int minPrice = INT_MAX, maxProfit = 0;
    for (int price : prices) {
        minPrice = min(minPrice, price); // Update the minimum price seen so far
        maxProfit = max(maxProfit, price - minPrice); // Calculate the maximum profit
    }
    return maxProfit;
}

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

    vector<int> prices(n);
    cout << "Enter the stock prices: ";
    for (int i = 0; i < n; i++) {
        cin >> prices[i];
    }

    cout << "Max profit: " << maxProfit(prices) << endl;
    return 0;
}
```

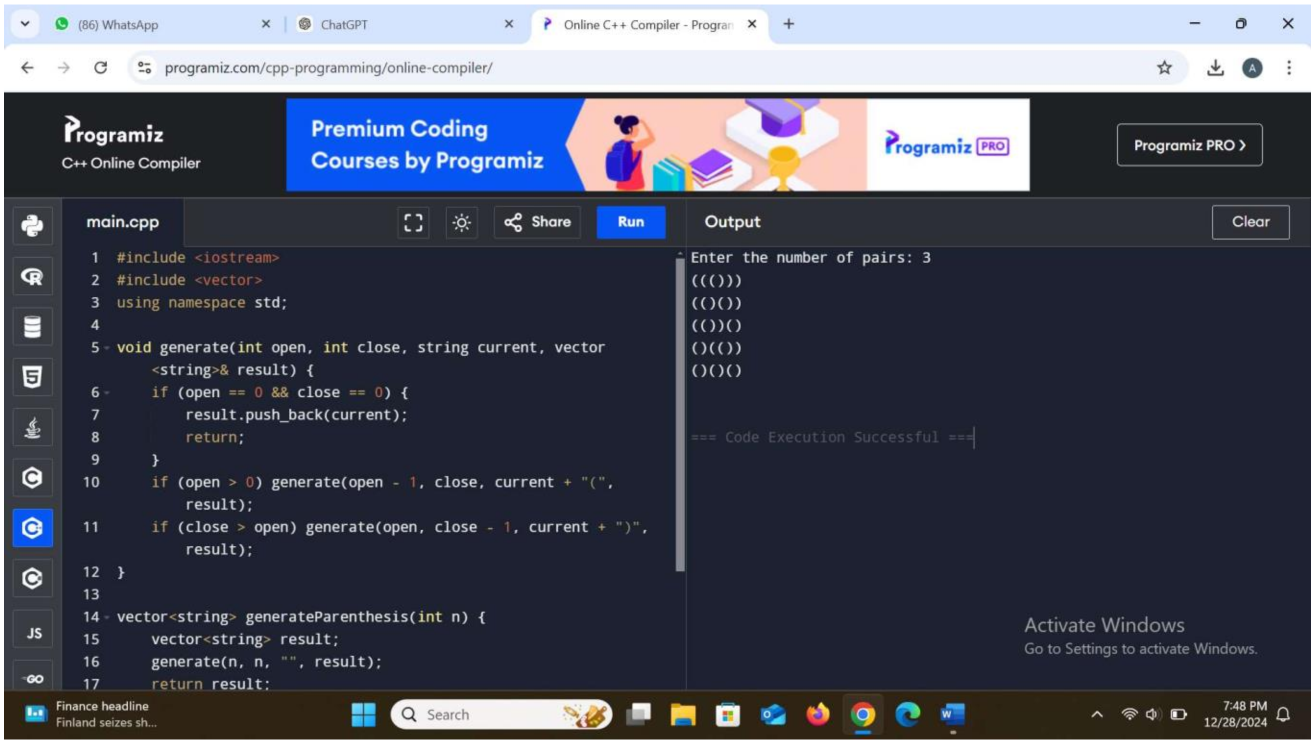


```
#include <iostream>
#include <vector>
#include <climits> // For INT_MAX
using namespace std;

int maxProfit(vector<int>& prices) {
    int minPrice = INT_MAX, maxProfit = 0;
    for (int price : prices) {
        minPrice = min(minPrice, price); // Update the minimum
            price seen so far
        maxProfit = max(maxProfit, price - minPrice); //
            Calculate the maximum profit
    }
    return maxProfit;
}

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

```
Enter the number of days: 6
Enter the stock prices: 7
1
5
3
6
4
Max profit: 5


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

5) Generate Parentheses

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

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

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

int main() {
    int n;
    cout << "Enter the number of pairs: ";
    cin >> n;
    vector<string> result = generateParenthesis(n);
    for (const string& s : result) {
        cout << s << endl;
    }
    return 0;
}
```

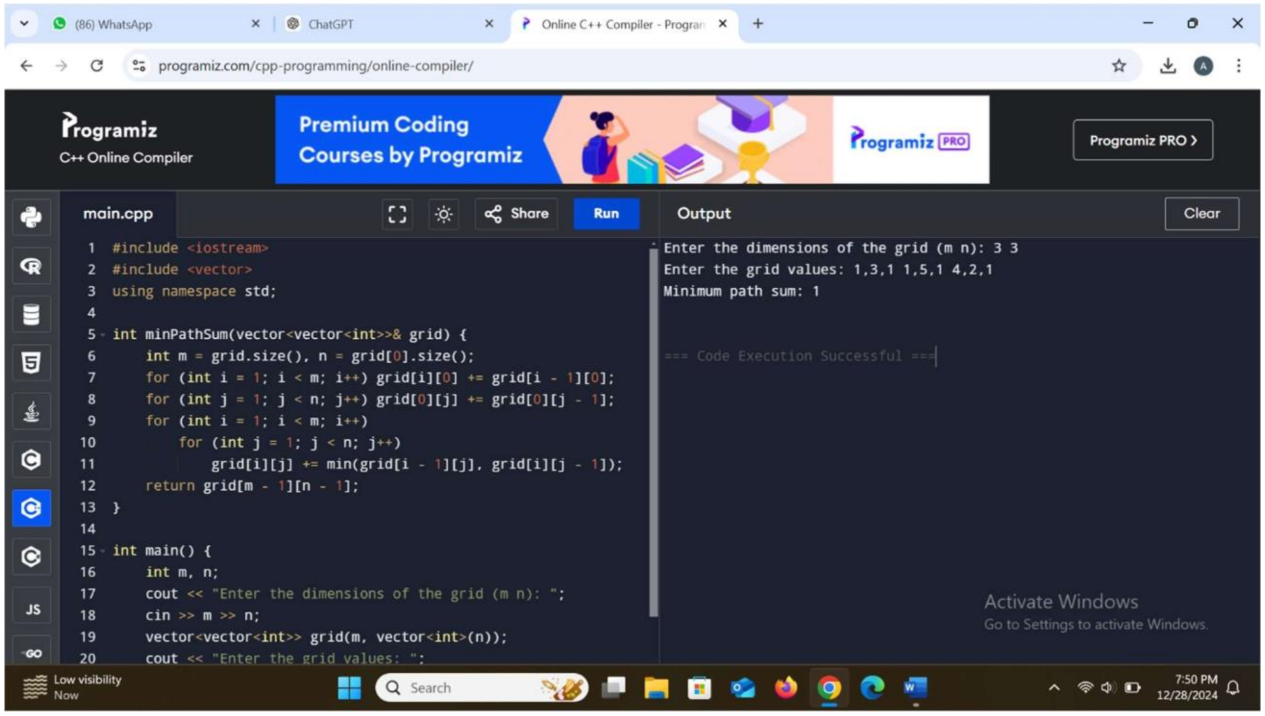


```
#include <iostream>
#include <vector>
using namespace std;

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

vector<string> generateParenthesis(int n) {
    vector<string> result;
    generate(n, n, "", result);
    return result;
```

```
Enter the number of pairs: 3
((()))
(()())
(())()
()(())
()()()

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

6) Minimum Path Sum

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

int minPathSum(vector<vector<int>>& grid) {
    int m = grid.size(), n = grid[0].size();
    for (int i = 1; i < m; i++) grid[i][0] += grid[i - 1][0];
    for (int j = 1; j < n; j++) grid[0][j] += grid[0][j - 1];
    for (int i = 1; i < m; i++)
        for (int j = 1; j < n; j++)
            grid[i][j] += min(grid[i - 1][j], grid[i][j - 1]);
    return grid[m - 1][n - 1];
}

int main() {
    int m, n;
    cout << "Enter the dimensions of the grid (m n): ";
    cin >> m >> n;
    vector<vector<int>> grid(m, vector<int>(n));
    cout << "Enter the grid values: ";
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> grid[i][j];
    cout << "Minimum path sum: " << minPathSum(grid) << endl;
    return 0;
}
```

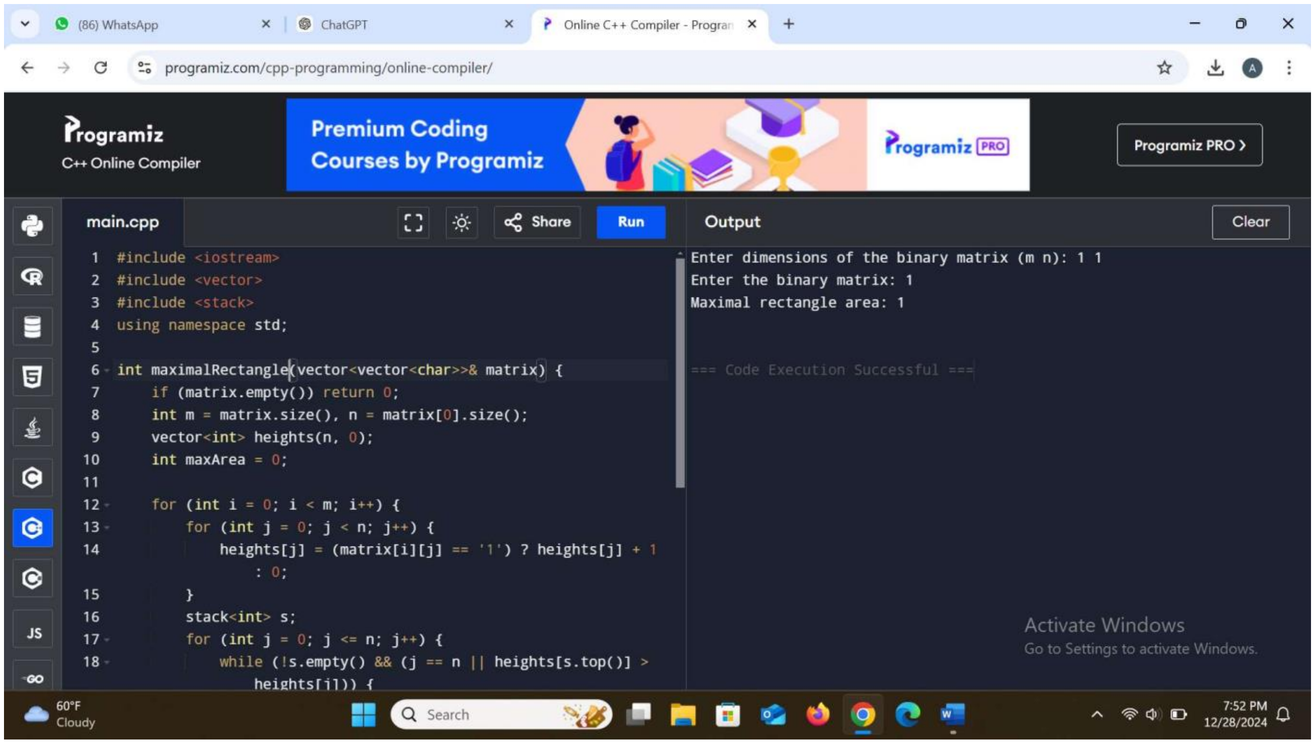


```
#include <iostream>
#include <vector>
using namespace std;

int minPathSum(vector<vector<int>>& grid) {
    int m = grid.size(), n = grid[0].size();
    for (int i = 1; i < m; i++) grid[i][0] += grid[i - 1][0];
    for (int j = 1; j < n; j++) grid[0][j] += grid[0][j - 1];
    for (int i = 1; i < m; i++)
        for (int j = 1; j < n; j++)
            grid[i][j] += min(grid[i - 1][j], grid[i][j - 1]);
    return grid[m - 1][n - 1];
}

int main() {
    int m, n;
    cout << "Enter the dimensions of the grid (m n): ";
    cin >> m >> n;
    vector<vector<int>> grid(m, vector<int>(n));
    cout << "Enter the grid values: ";
```

```
Enter the dimensions of the grid (m n): 3 3
Enter the grid values: 1,3,1 1,5,1 4,2,1
Minimum path sum: 1


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

7) Maximal Rectangle

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

int maximalRectangle(vector<vector<char>>& matrix) {
    if (matrix.empty()) return 0;
    int m = matrix.size(), n = matrix[0].size();
    vector<int> heights(n, 0);
    int maxArea = 0;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            heights[j] = (matrix[i][j] == '1') ? heights[j] + 1 : 0;
        }
        stack<int> s;
        for (int j = 0; j <= n; j++) {
            while (!s.empty() && (j == n || heights[s.top()] > heights[j])) {
                int h = heights[s.top()];
                s.pop();
                int width = s.empty() ? j : j - s.top() - 1;
                maxArea = max(maxArea, h * width);
            }
            s.push(j);
        }
    }
    return maxArea;
}

int main() {
    int m, n;
    cout << "Enter dimensions of the binary matrix (m n): ";
    cin >> m >> n;
    vector<vector<char>> matrix(m, vector<char>(n));
    cout << "Enter the binary matrix: ";
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> matrix[i][j];
    cout << "Maximal rectangle area: " << maximalRectangle(matrix) << endl;
    return 0;
}
```

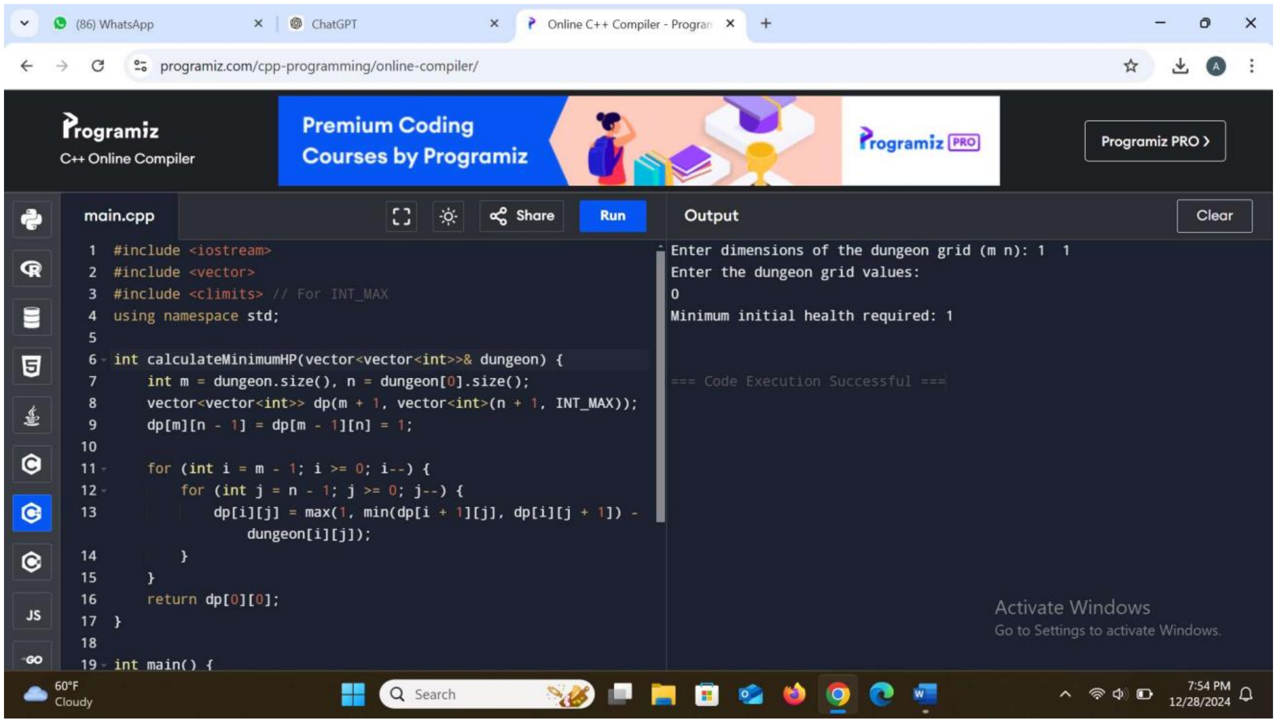


```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

int maximalRectangle(vector<vector<char>>& matrix) {
    if (matrix.empty()) return 0;
    int m = matrix.size(), n = matrix[0].size();
    vector<int> heights(n, 0);
    int maxArea = 0;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            heights[j] = (matrix[i][j] == '1') ? heights[j] + 1
                : 0;
        }
        stack<int> s;
        for (int j = 0; j <= n; j++) {
            while (!s.empty() && (j == n || heights[s.top()] >
                heights[j])) {
```

```
Enter dimensions of the binary matrix (m n): 1 1
Enter the binary matrix: 1
Maximal rectangle area: 1


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

8) Dungeon Game

```
#include <iostream>
#include <vector>
#include <climits> // For INT_MAX
using namespace std;

int calculateMinimumHP(vector<vector<int>>& dungeon) {
    int m = dungeon.size(), n = dungeon[0].size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, INT_MAX));
    dp[m][n - 1] = dp[m - 1][n] = 1;

    for (int i = m - 1; i >= 0; i--) {
        for (int j = n - 1; j >= 0; j--) {
            dp[i][j] = max(1, min(dp[i + 1][j], dp[i][j + 1]) - dungeon[i][j]);
        }
    }
    return dp[0][0];
}

int main() {
    int m, n;
    cout << "Enter dimensions of the dungeon grid (m n): ";
    cin >> m >> n;
    vector<vector<int>> dungeon(m, vector<int>(n));
    cout << "Enter the dungeon grid values: " << endl;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> dungeon[i][j];
        }
    }

    cout << "Minimum initial health required: " << calculateMinimumHP(dungeon) <<
endl;
    return 0;
}
```



(86) WhatsApp
ChatGPT
Online C++ Compiler - Progran
programiz.com/cpp-programming/online-compiler/
Programiz
C++ Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO >
main.cpp
Share
Run
Output
Clear

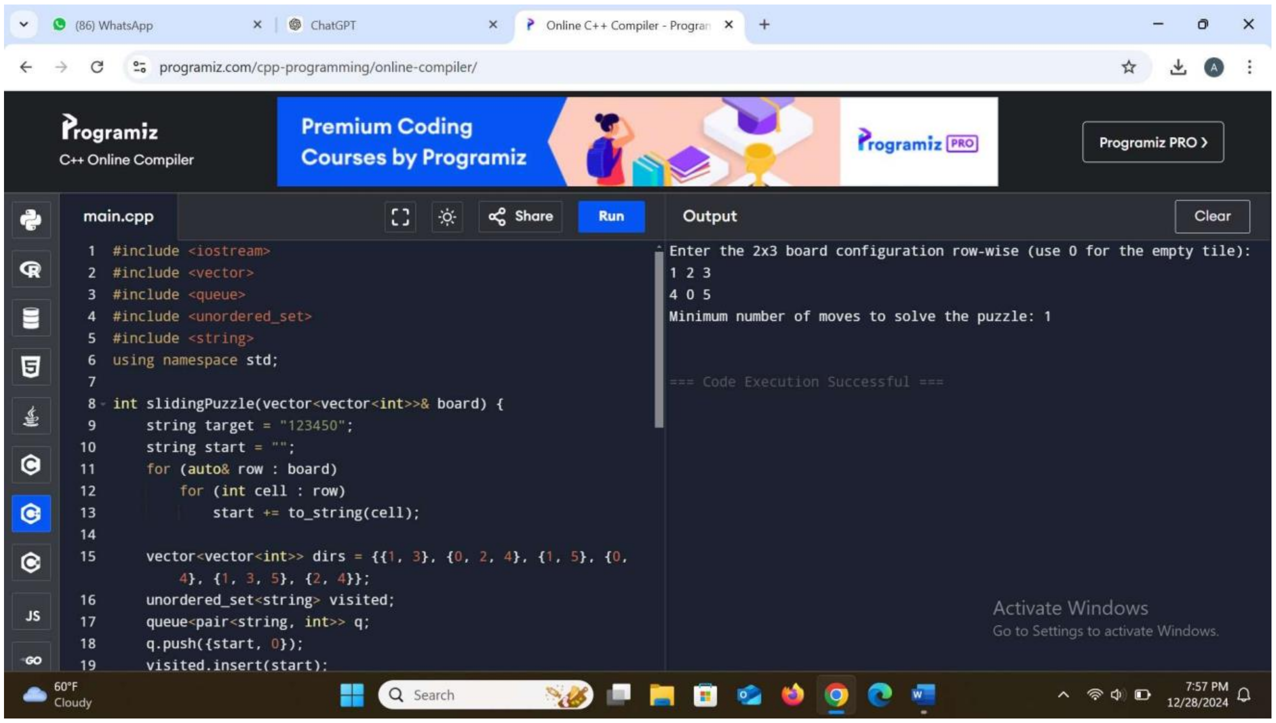
```
#include <iostream>
#include <vector>
#include <climits> // For INT_MAX
using namespace std;

int calculateMinimumHP(vector<vector<int>>& dungeon) {
    int m = dungeon.size(), n = dungeon[0].size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, INT_MAX));
    dp[m][n - 1] = dp[m - 1][n] = 1;

    for (int i = m - 1; i >= 0; i--) {
        for (int j = n - 1; j >= 0; j--) {
            dp[i][j] = max(1, min(dp[i + 1][j], dp[i][j + 1]) -
                dungeon[i][j]);
        }
    }
    return dp[0][0];
}

int main() {
```

```
Enter dimensions of the dungeon grid (m n): 1  1
Enter the dungeon grid values:
0
Minimum initial health required: 1


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

9) Sliding Puzzle

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>
#include <string>
using namespace std;

int slidingPuzzle(vector<vector<int>>& board) {
    string target = "123450";
    string start = "";
    for (auto& row : board)
        for (int cell : row)
            start += to_string(cell);

    vector<vector<int>> dirs = {{1, 3}, {0, 2, 4}, {1, 5}, {0, 4}, {1, 3, 5}, {2, 4}};
    unordered_set<string> visited;
    queue<pair<string, int>> q;
    q.push({start, 0});
    visited.insert(start);

    while (!q.empty()) {
        auto [curr, steps] = q.front();
        q.pop();
        if (curr == target) return steps;

        int zeroPos = curr.find('0');
        for (int dir : dirs[zeroPos]) {
            string next = curr;
            swap(next[zeroPos], next[dir]);
            if (visited.find(next) == visited.end()) {
                visited.insert(next);
                q.push({next, steps + 1});
            }
        }
    }
    return -1; // Impossible to solve
}
```

```
int main() {
    vector<vector<int>> board(2, vector<int>(3));
    cout << "Enter the 2x3 board configuration row-wise (use 0 for the empty tile):" <<
endl;
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 3; j++)
            cin >> board[i][j];

    int result = slidingPuzzle(board);
    if (result == -1) {
        cout << "The puzzle is unsolvable." << endl;
    } else {
        cout << "Minimum number of moves to solve the puzzle: " << result << endl;
    }
    return 0;
}
```

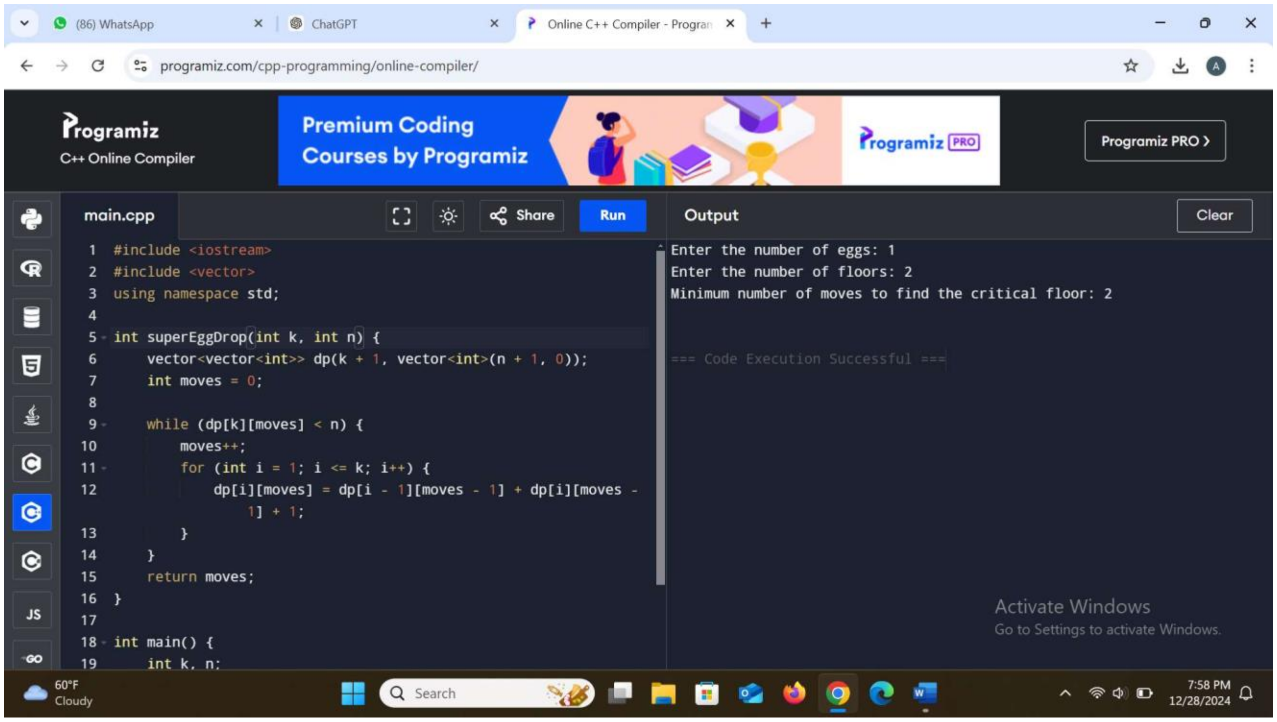


```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>
#include <string>
using namespace std;

int slidingPuzzle(vector<vector<int>>& board) {
    string target = "123450";
    string start = "";
    for (auto& row : board)
        for (int cell : row)
            start += to_string(cell);

    vector<vector<int>> dirs = {{1, 3}, {0, 2, 4}, {1, 5}, {0,
        4}, {1, 3, 5}, {2, 4}};
    unordered_set<string> visited;
    queue<pair<string, int>> q;
    q.push({start, 0});
    visited.insert(start);
```

```
Enter the 2x3 board configuration row-wise (use 0 for the empty tile):
1 2 3
4 0 5
Minimum number of moves to solve the puzzle: 1

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

10) Super Egg Drop

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

int superEggDrop(int k, int n) {
    vector<vector<int>> dp(k + 1, vector<int>(n + 1, 0));
    int moves = 0;

    while (dp[k][moves] < n) {
        moves++;
        for (int i = 1; i <= k; i++) {
            dp[i][moves] = dp[i - 1][moves - 1] + dp[i][moves - 1] + 1;
        }
    }
    return moves;
}

int main() {
    int k, n;
    cout << "Enter the number of eggs: ";
    cin >> k;
    cout << "Enter the number of floors: ";
    cin >> n;

    cout << "Minimum number of moves to find the critical floor: " <<
superEggDrop(k, n) << endl;
    return 0;
}
```



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

int superEggDrop(int k, int n) {
    vector<vector<int>> dp(k + 1, vector<int>(n + 1, 0));
    int moves = 0;

    while (dp[k][moves] < n) {
        moves++;
        for (int i = 1; i <= k; i++) {
            dp[i][moves] = dp[i - 1][moves - 1] + dp[i][moves -
                1] + 1;
        }
    }
    return moves;
}

int main() {
    int k, n;
```

```
Enter the number of eggs: 1
Enter the number of floors: 2
Minimum number of moves to find the critical floor: 2


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