AP-II Assignment

Name: Raja Babu **UID: 22BCS11971**
Section: 614-B

Q1. Set Matrix Zeroes Problem

Statement: Given an m x n matrix, if an element is 0, set its entire row and column to 0. The modification must be done in place without using additional storage for another matrix.

Example 1: Input: matrix = [[1, 1, 1], [1, 0, 1], [1, 1, 1]]

Output: [[1, 0, 1], [0, 0, 0], [1, 0, 1]]

Explanation: The element at position (1,1) is 0. Therefore, the entire row 1 and column 1 are set to 0.

Example 2: Input: matrix = [[0, 1, 2, 0], [3, 4, 5, 2], [1, 3, 1, 5]]
Output: [[0, 0, 0, 0], [0, 4, 5, 0], [0, 3, 1, 0]]

Explanation: The zeros in the first row (positions (0,0) and (0,3)) cause the entire first row and their corresponding columns to be set to 0.

Code:

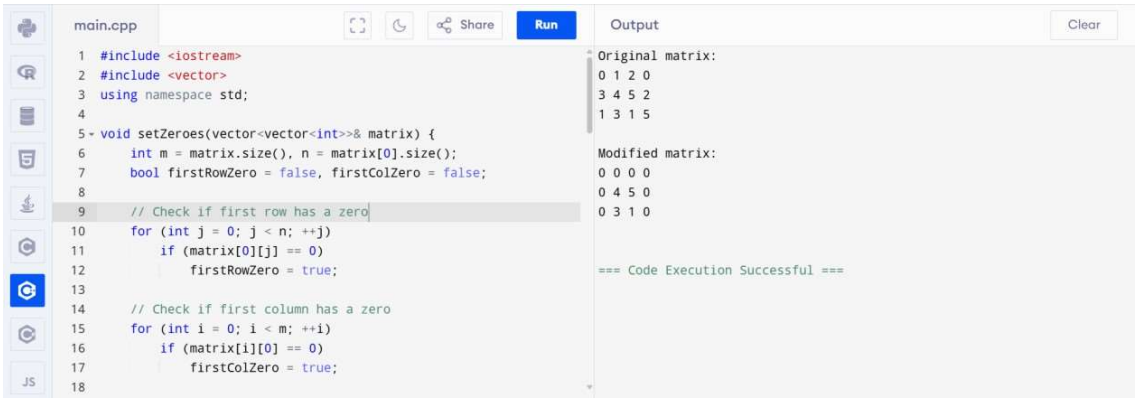
```
#include <iostream>

#include <vector>

using namespace std;


void setZeroes(vector<vector<int>>& matrix) {

    int m = matrix.size(), n = matrix[0].size();

    bool firstRowZero = false, firstColZero = false;


    // Check if first row has a zero
```

```
for (int j = 0; j < n; ++j)
    if (matrix[0][j] == 0)
        firstRowZero = true;

// Check if first column has a zero
for (int i = 0; i < m; ++i)
    if (matrix[i][0] == 0)
        firstColZero = true;

// Use first row and column as markers
for (int i = 1; i < m; ++i) {
    for (int j = 1; j < n; ++j) {
        if (matrix[i][j] == 0) {
            matrix[i][0] = 0;
            matrix[0][j] = 0;
        }
    }
}

// Set zeroes based on markers
for (int i = 1; i < m; ++i) {
    for (int j = 1; j < n; ++j) {
        if (matrix[i][0] == 0 || matrix[0][j] == 0)
            matrix[i][j] = 0;
    }
```

```
    }

    // Zero the first row if needed
    if (firstRowZero) {
        for (int j = 0; j < n; ++j)
            matrix[0][j] = 0;
    }

    // Zero the first column if needed
    if (firstColZero) {
        for (int i = 0; i < m; ++i)
            matrix[i][0] = 0;
    }
}

void printMatrix(const vector<vector<int>>& matrix) {
    for (const auto& row : matrix) {
        for (int val : row)
            cout << val << " ";
        cout << endl;
    }
}

int main() {
    vector<vector<int>> matrix = {
```

```
        {0, 1, 2, 0},
        {3, 4, 5, 2},
        {1, 3, 1, 5}
    };

    cout << "Original matrix:\n";
    printMatrix(matrix);

    setZeroes(matrix);

    cout << "\nModified matrix:\n";
    printMatrix(matrix);

    return 0;
}
```

Output:



Q2. Longest Substring Without Repeating Characters

Problem Statement: Given a string s, find the length of the longest substring that does not contain any repeating characters.

Example 1: Input: s = "abcabcbb" Output: 3

Explanation: The longest substring without repeating characters is "abc", which has a length of 3.

Example 2: Input: s = "bbbbb" Output: 1

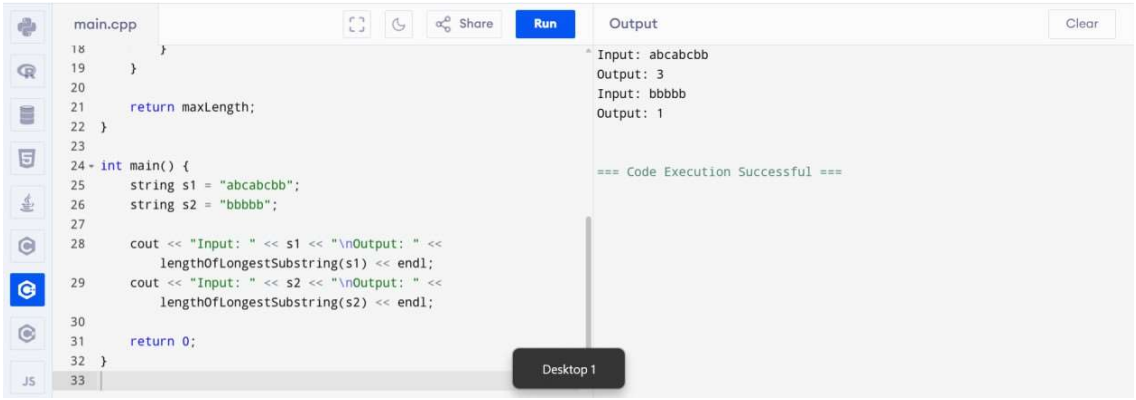
Explanation: All characters in the string are the same, so the longest substring with unique characters is "b", with a length of 1.

Code:

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

int lengthOfLongestSubstring(string s) {
    unordered_set<char> seen;
    int left = 0, right = 0, maxLength = 0;

    while (right < s.length()) {
        if (seen.find(s[right]) == seen.end()) {
            seen.insert(s[right]);
            maxLength = max(maxLength, right - left + 1);
            right++;
```

```
        } else {
            seen.erase(s[left]);
            left++;
        }
    }
    return maxLength;
}
int main() {
    string s1 = "abcabcbb";
    string s2 = "bbbbb";
    cout << "Input: " << s1 << "\nOutput: " <<
lengthOfLongestSubstring(s1) << endl;
    cout << "Input: " << s2 << "\nOutput: " <<
lengthOfLongestSubstring(s2) << endl;

    return 0;
}
```

Output:



Q3. Reverse Linked List II Problem

Statement: Given the head of a singly linked list and two integers left and right, reverse the nodes of the list from position left to right, and return the modified list.

Example 1: Input: Linked list: [1, 2, 3, 4, 5]; left = 2; right = 4

Output: [1, 4, 3, 2, 5]

Explanation: The sublist from position 2 to 4 ([2, 3, 4]) is reversed to become [4, 3, 2], while the rest of the list remains unchanged.

Example 2: Input: Linked list: [1, 2, 3, 4, 5]; left = 1; right = 5

Output: [5, 4, 3, 2, 1]

Explanation: The entire list is reversed because the reversal starts at the first node and ends at the last node

Code:

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

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to reverse the sublist from position left to right
```

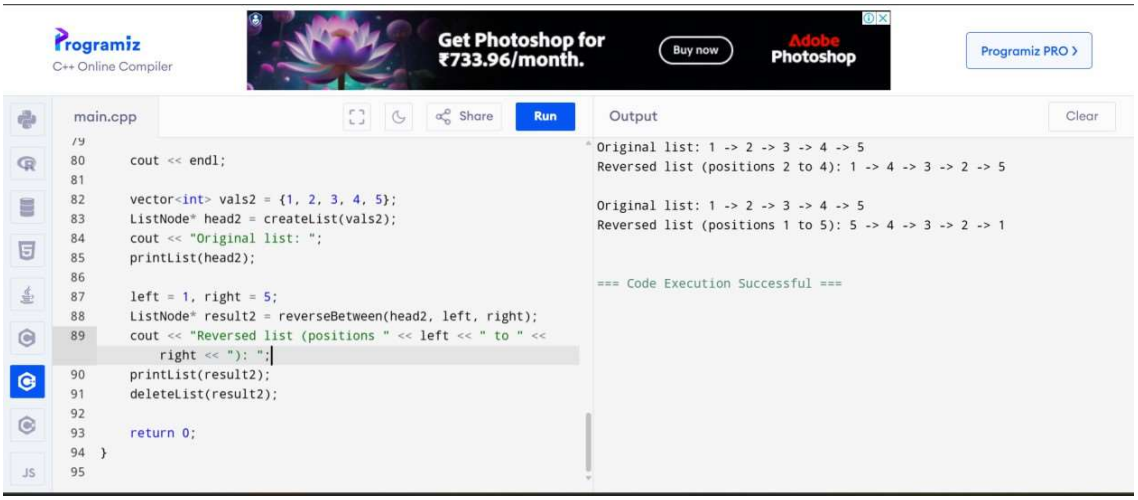
```
ListNode* reverseBetween(ListNode* head, int left, int right) {
    if (!head || left == right) return head;

    ListNode dummy(0);
    dummy.next = head;
    ListNode* prev = &dummy;

    // Move prev to the node before 'left'
    for (int i = 1; i < left; ++i) {
        prev = prev->next;
    }

    // Reverse the sublist
    ListNode* curr = prev->next;
    for (int i = 0; i < right - left; ++i) {
        ListNode* temp = curr->next;
        curr->next = temp->next;
        temp->next = prev->next;
        prev->next = temp;
    }

    return dummy.next;
}

// Helper to print the linked list
```

```
void printList(ListNode* head) {
    while (head) {
        cout << head->val;
        if (head->next) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}

// Helper to create a linked list from vector
ListNode* createList(const vector<int>& vals) {
    if (vals.empty()) return nullptr;
    ListNode* head = new ListNode(vals[0]);
    ListNode* curr = head;
    for (size_t i = 1; i < vals.size(); ++i) {
        curr->next = new ListNode(vals[i]);
        curr = curr->next;
    }
    return head;
}

// Helper to delete the list and free memory
void deleteList(ListNode* head) {
    while (head) {
        ListNode* temp = head;
```

```
        head = head->next;
        delete temp;
    }
}

int main() {
    vector<int> vals1 = {1, 2, 3, 4, 5};
    ListNode* head1 = createList(vals1);
    cout << "Original list: ";
    printList(head1);

    int left = 2, right = 4;
    ListNode* result1 = reverseBetween(head1, left, right);
    cout << "Reversed list (positions " << left << " to " << right << "):
";
    printList(result1);
    deleteList(result1);

    cout << endl;

    vector<int> vals2 = {1, 2, 3, 4, 5};
    ListNode* head2 = createList(vals2);
    cout << "Original list: ";
    printList(head2);
```

```
    left = 1, right = 5;
    ListNode* result2 = reverseBetween(head2, left, right);
    cout << "Reversed list (positions " << left << " to " << right << "):
";
    printList(result2);
    deleteList(result2);


    return 0;
}
```

Output:



Q4. Detect a Cycle in a Linked List

Problem Statement: Given the head of a linked list, determine whether the linked list contains a cycle. A cycle occurs if a node's next pointer points to a previous node in the list.

Example 1: Input: Linked list: [3, 2, 0, -4] with the tail node (-4) pointing to the node with value 2.

Output: true Explanation: The tail node connects back to an earlier node, forming a cycle.

Example 2: Input: Linked list: [1, 2] with no cycle (each node points to null at the end).

Output: false

Explanation: There is no cycle since no node points back to a previous node.

Code:

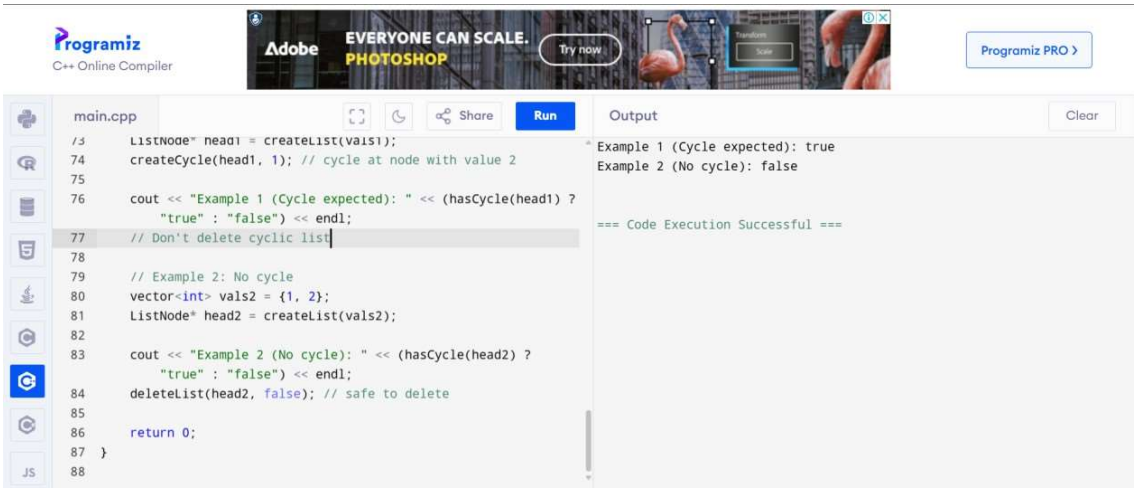
```
#include <iostream>
#include <vector>
using namespace std;

// Definition for singly-linked list
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to detect a cycle in the linked list using Floyd's Algorithm
bool hasCycle(ListNode* head) {
    if (!head || !head->next) return false;

    ListNode* slow = head;
    ListNode* fast = head;
```

```
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;

        if (slow == fast) return true;
    }

    return false;
}

// Helper to create a linked list from a vector (no cycle)
ListNode* createList(const vector<int>& vals) {
    if (vals.empty()) return nullptr;
    ListNode* head = new ListNode(vals[0]);
    ListNode* curr = head;
    for (size_t i = 1; i < vals.size(); ++i) {
        curr->next = new ListNode(vals[i]);
        curr = curr->next;
    }
    return head;
}

// Helper to create a cycle in the list at the given position (0-based
index)
```

```
void createCycle(ListNode* head, int pos) {
    if (pos < 0) return;

    ListNode* cycleNode = nullptr;
    ListNode* curr = head;
    int index = 0;

    while (curr->next) {
        if (index == pos) cycleNode = curr;
        curr = curr->next;
        index++;
    }

    curr->next = cycleNode; // create the cycle
}

// Memory leak warning: Skipping deletion of cyclic list to avoid crash
void deleteList(ListNode* head, bool has_cycle) {
    if (has_cycle) return;

    while (head) {
        ListNode* temp = head;
        head = head->next;
        delete temp;
```

```
    }
}

// Main function to test both cases
int main() {
    // Example 1: Cycle exists
    vector<int> vals1 = {3, 2, 0, -4};
    ListNode* head1 = createList(vals1);
    createCycle(head1, 1); // cycle at node with value 2

    cout << "Example 1 (Cycle expected): " << (hasCycle(head1) ? "true" : "false") << endl;
    // Don't delete cyclic list

    // Example 2: No cycle
    vector<int> vals2 = {1, 2};
    ListNode* head2 = createList(vals2);

    cout << "Example 2 (No cycle): " << (hasCycle(head2) ? "true" : "false") << endl;
    deleteList(head2, false); // safe to delete

    return 0;
}
```

Output:-



Q5. The Skyline Problem

Problem Statement: Given a list of buildings represented as [left, right, height], where each building is a rectangle, return the key points of the skyline. A key point is represented as [x, y], where x is the x-coordinate where the height changes to y.

Example 1: Input: buildings = [[2, 9, 10], [3, 7, 15], [5, 12, 12], [15, 20, 10], [19, 24, 8]]

Output: [[2, 10], [3, 15], [7, 12], [12, 0], [15, 10], [20, 8], [24, 0]]
Explanation: The skyline starts at x = 2 with height 10, rises to height 15 at x = 3, drops to 12 at x = 7, then to 0 at x = 12, rises again at x = 15, changes at x = 20, and finally drops to 0 at x = 24.

Example 2: Input: buildings = [[0, 2, 3], [2, 5, 3]]

Output: [[0, 3], [5, 0]]

Explanation: Both buildings have the same height of 3. The skyline begins at x = 0 with height 3 and ends at x = 5 when the building ends, resulting in a drop to 0.

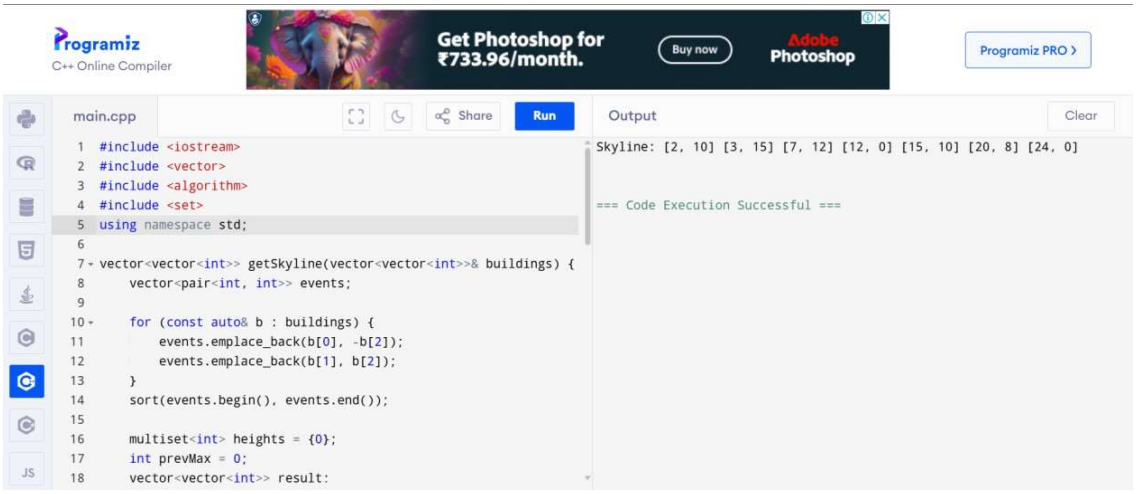
Code:

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

vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
    vector<pair<int, int>> events;

    for (const auto& b : buildings) {
        events.emplace_back(b[0], -b[2]);
        events.emplace_back(b[1], b[2]);
    }
    sort(events.begin(), events.end());

    multiset<int> heights = {0};
    int prevMax = 0;
    vector<vector<int>> result;

    for (auto [x, h] : events) {
        if (h < 0) {
            heights.insert(-h);
        } else {
            heights.erase(heights.find(h));
```

```
        }

        int currMax = *heights.rbegin();
        if (currMax != prevMax) {
            result.push_back({x, currMax});
            prevMax = currMax;
        }
    }

    return result;
}

void printSkyline(const vector<vector<int>>& skyline) {
    for (const auto& point : skyline) {
        cout << "[" << point[0] << ", " << point[1] << "] ";
    }
    cout << endl;
}

int main() {
    vector<vector<int>> buildings = {
        {2, 9, 10},
        {3, 7, 15},
        {5, 12, 12},
        {15, 20, 10},
```

```
        {19, 24, 8}
    };

    vector<vector<int>> skyline = getSkyline(buildings);
    cout << "Skyline: ";
    printSkyline(skyline);

    return 0;
}
```

Output:



Q6. Longest Increasing Subsequence II

Problem Statement: Given an integer array nums, find the length of the longest strictly increasing subsequence. A subsequence is derived from the array by deleting some or no elements without changing the order of the remaining elements.

Example 1: Input: nums = [10, 9, 2, 5, 3, 7, 101, 18]

Output: 4

Explanation: One longest increasing subsequence is [2, 3, 7, 101], which has a length of 4.

Example 2: Input: nums = [0, 1, 0, 3, 2, 3]

Output: 4

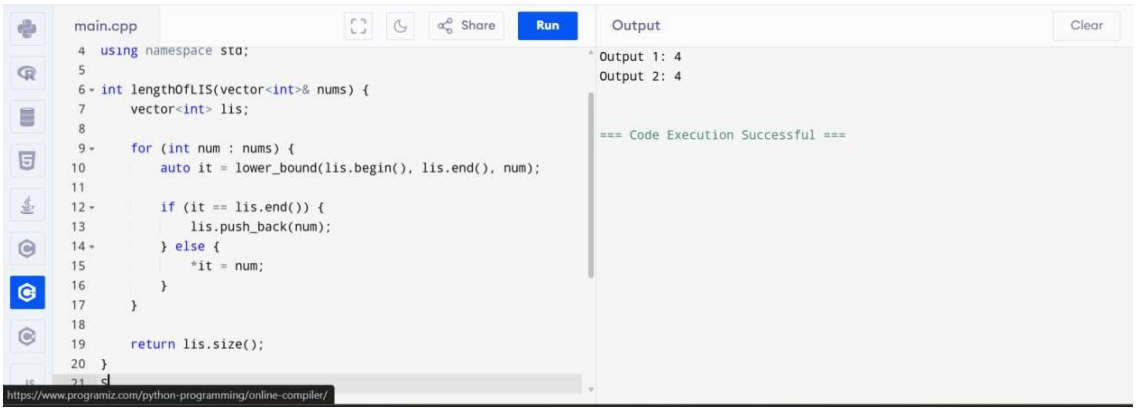
Explanation: One valid subsequence is [0, 1, 2, 3] with a length of 4.

Code:

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

int lengthOfLIS(vector<int>& nums) {
    vector<int> lis;

    for (int num : nums) {
        auto it = lower_bound(lis.begin(), lis.end(), num);

        if (it == lis.end()) {
            lis.push_back(num);
        } else {
            *it = num;
        }
    }
```

```
    return lis.size();
}

int main() {
    vector<int> nums1 = {10, 9, 2, 5, 3, 7, 101, 18};
    cout << "Output 1: " << lengthOfLIS(nums1) << endl;

    vector<int> nums2 = {0, 1, 0, 3, 2, 3};
    cout << "Output 2: " << lengthOfLIS(nums2) << endl;

    return 0;
}
```

Output:



Q7. Search a 2D Matrix II

Problem Statement: Given an m x n matrix where each row is sorted in ascending order from left to right and each column is sorted in

ascending order from top to bottom, and an integer target, determine if the target exists in the matrix.

Example 1: Input: matrix = [[1, 4, 7, 11, 15], [2, 5, 8, 12, 19], [3, 6, 9, 16, 22], [10, 13, 14, 17, 24], [18, 21, 23, 26, 30]], target = 5

Output: true

Explanation: The target value 5 is found in the matrix at the second row, second column.

Example 2: Input: matrix = [[1, 4, 7, 11, 15], [2, 5, 8, 12, 19], [3, 6, 9, 16, 22], [10, 13, 14, 17, 24], [18, 21, 23, 26, 30]], target = 20

Output: false

Explanation: The target value 20 is not present anywhere in the matrix.

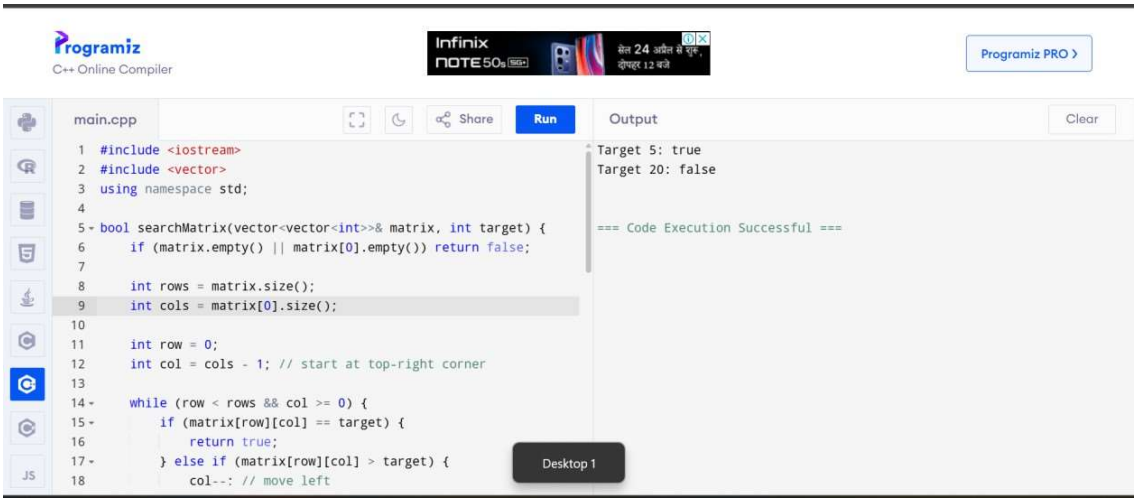
Code:

```
#include <iostream>

#include <vector>

using namespace std;


bool searchMatrix(vector<vector<int>>& matrix, int target) {

    if (matrix.empty() || matrix[0].empty()) return false;


    int rows = matrix.size();

    int cols = matrix[0].size();


    int row = 0;

    int col = cols - 1; // start at top-right corner
```

```
    while (row < rows && col >= 0) {
        if (matrix[row][col] == target) {
            return true;
        } else if (matrix[row][col] > target) {
            col--; // move left
        } else {
            row++; // move down
        }
    }

    return false;
}

// Test the code
int main() {
    vector<vector<int>> matrix = {
        {1, 4, 7, 11, 15},
        {2, 5, 8, 12, 19},
        {3, 6, 9, 16, 22},
        {10, 13, 14, 17, 24},
        {18, 21, 23, 26, 30}
    };

    int target1 = 5;
    int target2 = 20;
```

```
    cout << "Target " << target1 << ": " << (searchMatrix(matrix, target1) ? "true" : "false") << endl;
    cout << "Target " << target2 << ": " << (searchMatrix(matrix, target2) ? "true" : "false") << endl;

    return 0;
}
```

Output:



Q8. Word Break Problem

Statement: Given a string s and a dictionary wordDict containing a list of words, determine if s can be segmented into a space-separated sequence of one or more dictionary words. The same word can be reused multiple times.

Example 1: Input: s = "leetcode", wordDict = ["leet", "code"]

Output: true

Explanation: The string "leetcode" can be segmented as "leet code", where both "leet" and "code" are present in the dictionary.

Example 2: Input: s = "applepenapple", wordDict = ["apple", "pen"]
Output: true

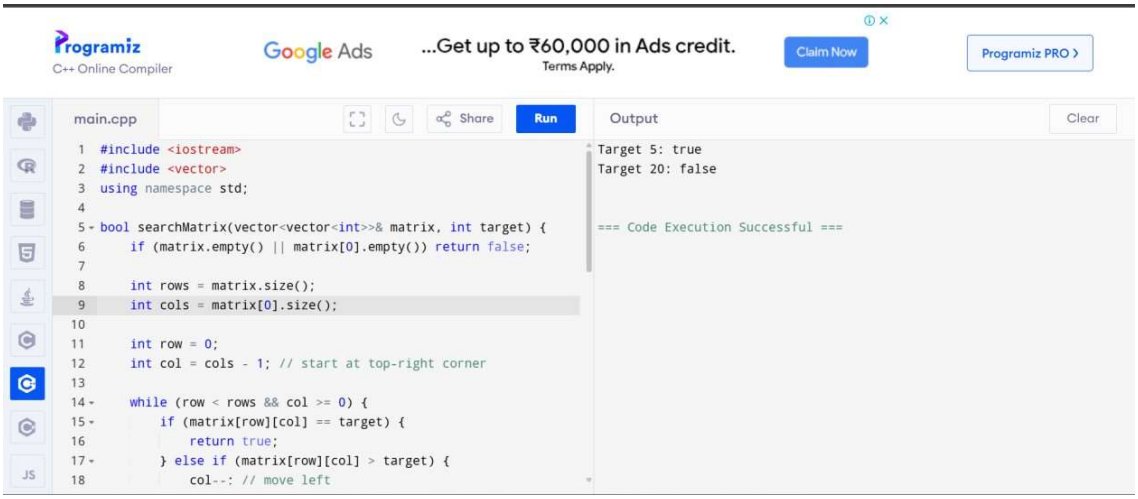
Explanation: The string can be segmented as "apple pen apple". Reusing "apple" is allowed, and both words exist in the dictionary.

Code:

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

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) return false;

    int rows = matrix.size();
    int cols = matrix[0].size();

    int row = 0;
    int col = cols - 1; // start at top-right corner

    while (row < rows && col >= 0) {
        if (matrix[row][col] == target) {
            return true;
        } else if (matrix[row][col] > target) {
            col--; // move left
        } else {
```

```
            row++; // move down
        }
    }

    return false;
}
int main() {
    vector<vector<int>> matrix = {
        {1, 4, 7, 11, 15},
        {2, 5, 8, 12, 19},
        {3, 6, 9, 16, 22},
        {10, 13, 14, 17, 24},
        {18, 21, 23, 26, 30}
    };
    int target1 = 5;
    int target2 = 20;
    cout << "Target " << target1 << ": " << (searchMatrix(matrix, target1) ? "true" : "false") << endl;
    cout << "Target " << target2 << ": " << (searchMatrix(matrix, target2) ? "true" : "false") << endl;

    return 0;
}
```

Output:



Q9. Longest Increasing Path in a Matrix Problem

Statement: Given an m x n integer matrix, find the length of the longest strictly increasing path. You can move up, down, left, or right from each cell. Diagonal moves and moves outside the boundaries are not allowed.

Example 1: Input: matrix = [[9, 9, 4], [6, 6, 8], [2, 1, 1]]

Output: 4

Explanation: One of the longest increasing paths is [1, 2, 6, 9]. Starting from the bottomleft cell, move to 2, then 6, and finally to 9.

Example 2: Input: matrix = [[3, 4, 5], [3, 2, 6], [2, 2, 1]]

Output: 4

Explanation: A valid longest increasing path is [3, 4, 5, 6]. The path moves strictly upward, increasing in value at each step.

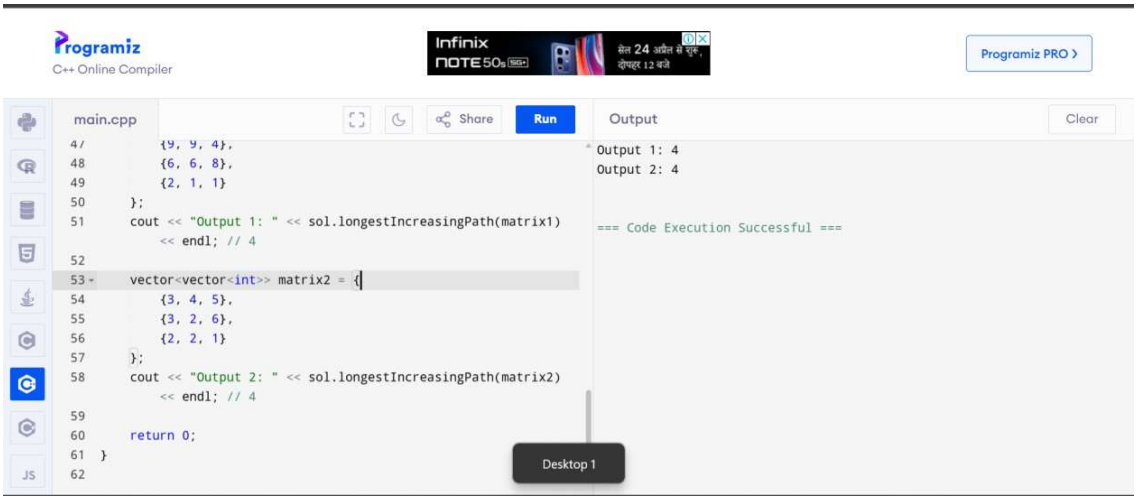
Code:

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

class Solution {
public:
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        if (matrix.empty()) return 0;

        int m = matrix.size();
        int n = matrix[0].size();
        vector<vector<int>> dp(m, vector<int>(n, 0));
        int maxLen = 0;

        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
                maxLen = max(maxLen, dfs(matrix, dp, i, j));

        return maxLen;
    }

private:
    vector<pair<int, int>> directions = {{0,1}, {1,0}, {0,-1}, {-1,0}};
```

```
    int dfs(vector<vector<int>>& matrix, vector<vector<int>>& dp, int
i, int j) {
        if (dp[i][j] != 0) return dp[i][j];

        int maxPath = 1;
        for (auto [dx, dy] : directions) {
            int x = i + dx, y = j + dy;
            if (x >= 0 && x < matrix.size() && y >= 0 && y <
matrix[0].size()
                && matrix[x][y] > matrix[i][j]) {
                maxPath = max(maxPath, 1 + dfs(matrix, dp, x, y));
            }
        }

        return dp[i][j] = maxPath;
    }
};

// Test the solution
int main() {
    Solution sol;

    vector<vector<int>> matrix1 = {
        {9, 9, 4},
```

```
        {6, 6, 8},
        {2, 1, 1}
    };
    cout << "Output 1: " << sol.longestIncreasingPath(matrix1) <<
endl; // 4
    vector<vector<int>> matrix2 = {
        {3, 4, 5},
        {3, 2, 6},
        {2, 2, 1}
    };
    cout << "Output 2: " << sol.longestIncreasingPath(matrix2) <<
endl; // 4
  return 0;
}
```

Output:



Q10. Trapping Rain Water Problem

Statement: Given n non-negative integers representing an elevation map where the width of each bar is 1, compute the total amount of water that can be trapped after raining.

Example 1: Input: height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

Output: 6

Explanation: Water is trapped between the bars. For example, water accumulates in the gaps between the bars at heights 1 and 2, resulting in a total of 6 units of water.

Example 2: Input: height = [4, 2, 0, 3, 2, 5]

Output: 9

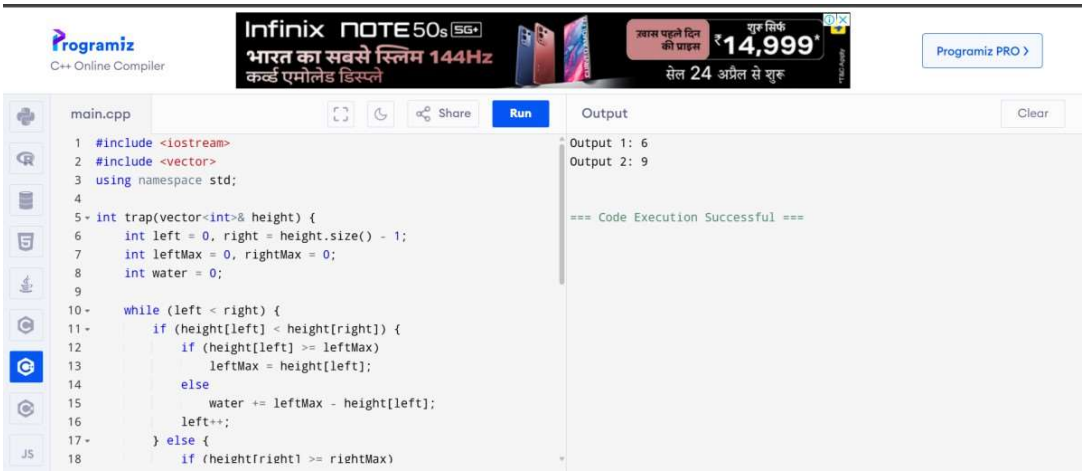
Explanation: The elevation map forms valleys where water is trapped. Calculations at each valley sum up to 9 units of water in total.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int trap(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int leftMax = 0, rightMax = 0;
    int water = 0;
    while (left < right) {
        if (height[left] < height[right]) {
            if (height[left] >= leftMax)
                leftMax = height[left];
            else
```

```
            water += leftMax - height[left];
        left++;
    } else {
        if (height[right] >= rightMax)
            rightMax = height[right];
        else
            water += rightMax - height[right];
        right--;
    }
  }
  return water;
}
int main() {
  vector<int> height1 = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
  cout << "Output 1: " << trap(height1) << endl; // 6
  vector<int> height2 = {4, 2, 0, 3, 2, 5};
  cout << "Output 2: " << trap(height2) << endl; // 9
  return 0;
}
```

Output:


Programiz
C++ Online Compiler
Infinix NOTE 50s 5G+
भारत का सबसे स्लिम 144Hz
कर्व्ड एमोलेड डिस्प्ले
₹14,999*
सेल 24 अप्रैल से शुरू
Programiz PRO
main.cpp
Share
Run
Output
Clear
#include <iostream>
#include <vector>
using namespace std;
int trap(vector<int>& height) {
int left = 0, right = height.size() - 1;
int leftMax = 0, rightMax = 0;
int water = 0;
while (left < right) {
if (height[left] < height[right]) {
if (height[left] >= leftMax)
leftMax = height[left];
else
water += leftMax - height[left];
left++;
} else {
Output 1: 6
Output 2: 9
=== Code Execution Successful ===