Assignment- 9

Submitted By: Ashish kumar singh

Class: IOT_614(B)

UID: 22BCS16892

Q1. Set Matrix Zeroes

Problem:
Given an m x n integer matrix, if an element is 0, set its entire row and column to 0's.

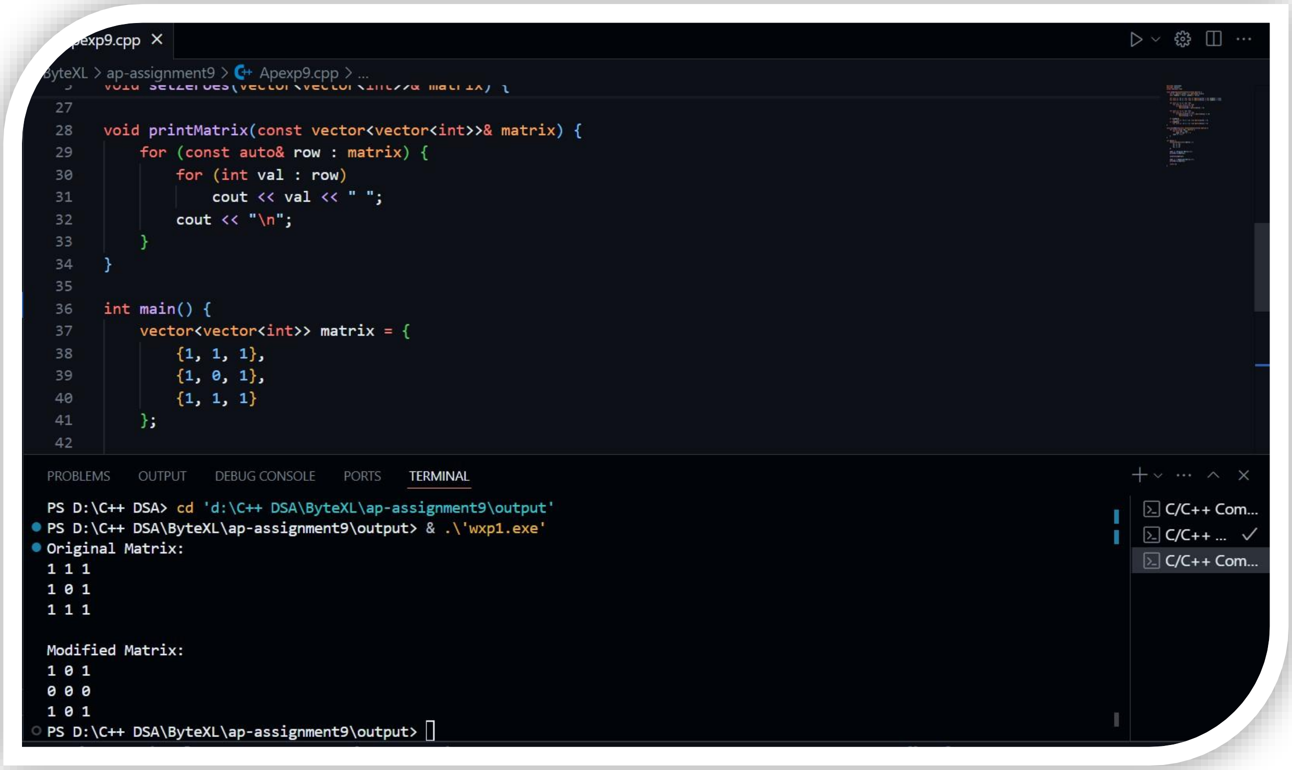
Approach:
Use first row and first column as markers. Use extra variables to track if first row/column should be zeroed.

Time Complexity: O(m * n)
Space Complexity: O(1)

Solution:

```
#include <iostream>

#include <vector>

using namespace std;


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

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

    bool rowZero = false, colZero = false;


    for (int i = 0; i < m; ++i) if (matrix[i][0] == 0) colZero = true;

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


    for (int i = 1; i < m; ++i)

        for (int j = 1; j < n; ++j)

            if (matrix[i][j] == 0)
```

```
                matrix[i][0] = matrix[0][j] = 0;

    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;

    if (colZero)
        for (int i = 0; i < m; ++i) matrix[i][0] = 0;
    if (rowZero)
        for (int j = 0; j < n; ++j) matrix[0][j] = 0;
}

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

int main() {
    vector<vector<int>> matrix = {
        {1, 1, 1},
        {1, 0, 1},
        {1, 1, 1}
    };
    cout << "Original Matrix:\n";
    printMatrix(matrix);


    setZeroes(matrix);


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


    return 0;
}
```



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

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

int main() {
    vector<vector<int>> matrix = {
        {1, 1, 1},
        {1, 0, 1},
        {1, 1, 1}
    };

```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'wxp1.exe'
Original Matrix:
1 1 1
1 0 1
1 1 1

Modified Matrix:
1 0 1
0 0 0
1 0 1
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q2. Longest Substring Without Repeating Characters

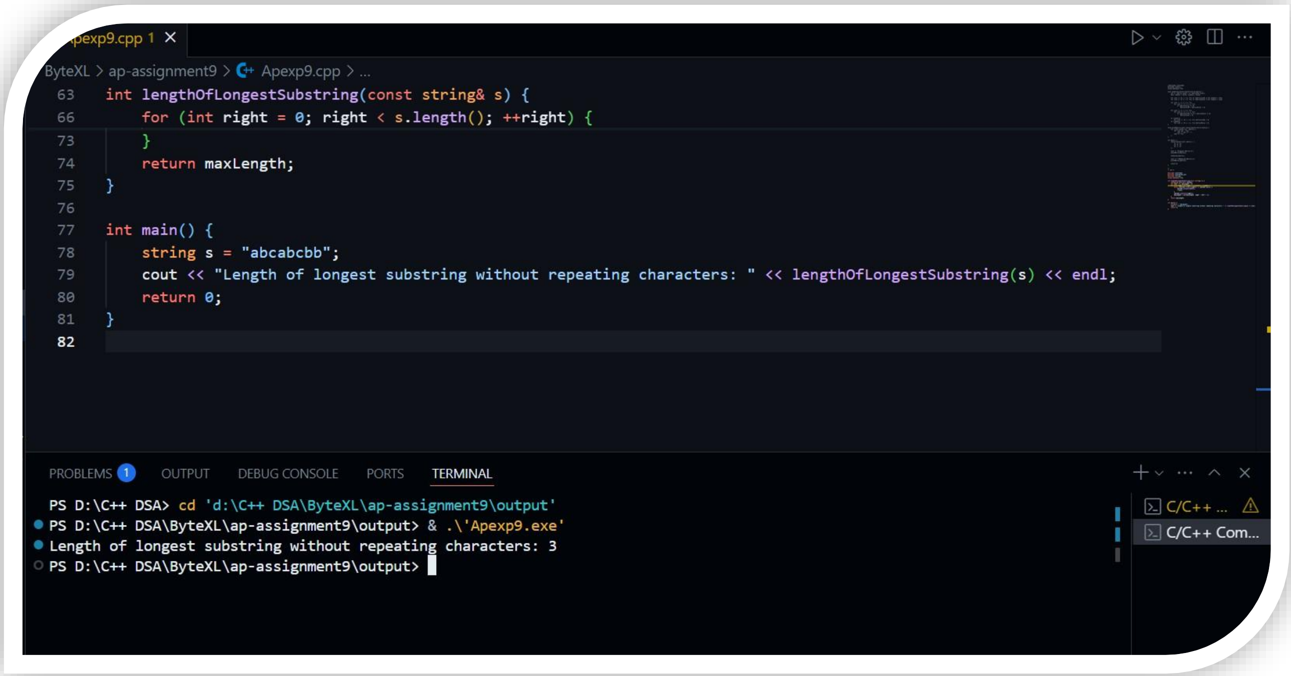
Problem:
Given a string s, find the length of the longest substring without repeating characters.

Approach:
Use a sliding window with a hashmap to store last seen index of characters.

Time Complexity: O(n)
Space Complexity: O(256)

Solution:

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

int lengthOfLongestSubstring(const string& s) {
    unordered_set<char> charSet;
    int left = 0, maxLength = 0;
    for (int right = 0; right < s.length(); ++right) {
        while (charSet.find(s[right]) != charSet.end()) {
            charSet.erase(s[left]);
            ++left;
        }
        charSet.insert(s[right]);
        maxLength = max(maxLength, right - left + 1);
    }
    return maxLength;
}

int main() {
    string s = "abcabcbb";
    cout << "Length of longest substring without repeating characters: " << lengthOfLongestSubstring(s) << endl;

    return 0;

}
```



```
int lengthOfLongestSubstring(const string& s) {
    for (int right = 0; right < s.length(); ++right) {
    }
    return maxLength;
}

int main() {
    string s = "abcabcbb";
    cout << "Length of longest substring without repeating characters: " << lengthOfLongestSubstring(s) << endl;
    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Length of longest substring without repeating characters: 3
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q3. Reverse Linked List II

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.

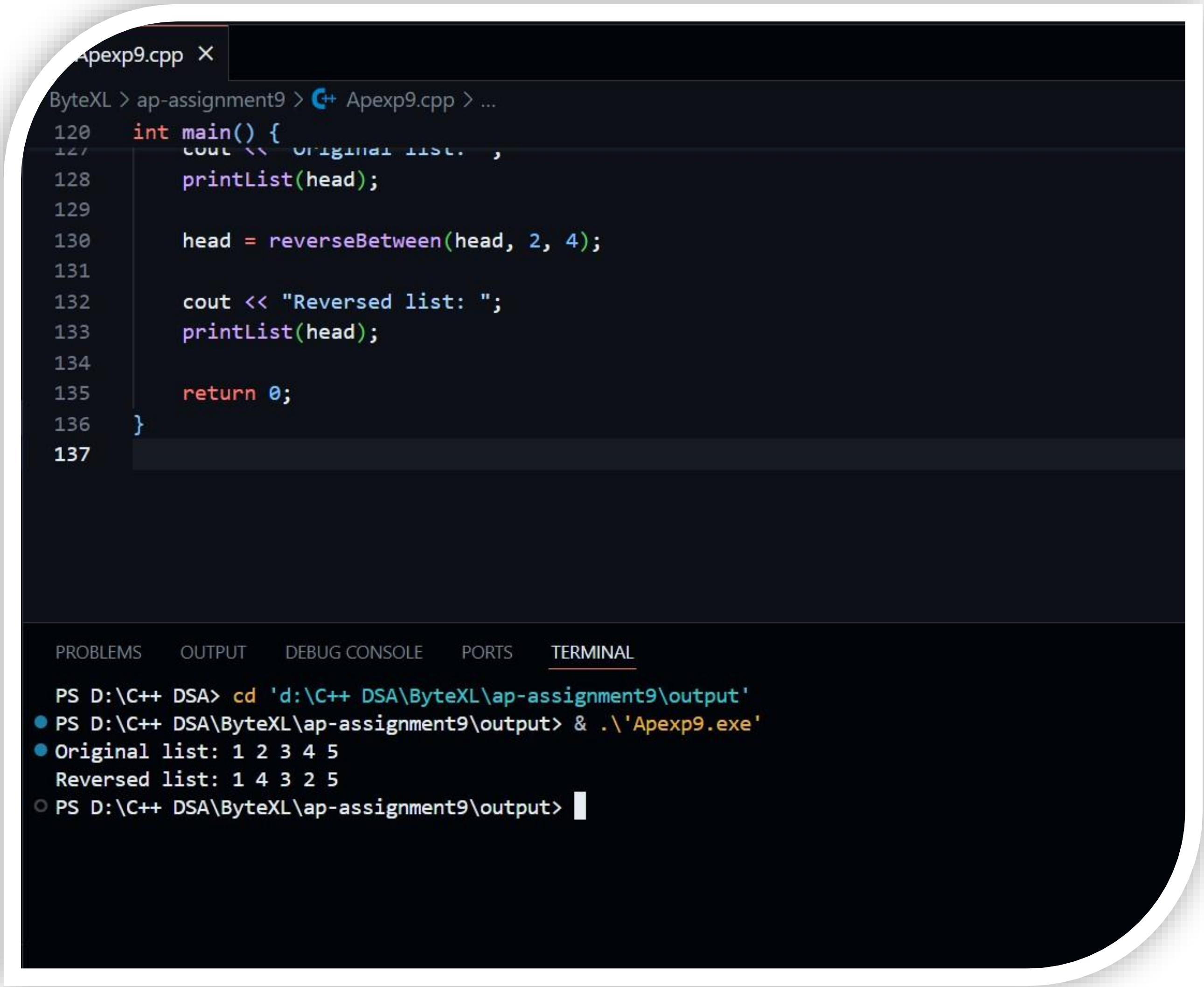
Approach:
Reverse the sublist by keeping track of pointers.

Time Complexity: O(n)
Space Complexity: O(1)

Solution:

```
#include <iostream>

using namespace std;
```

```
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* reverseBetween(ListNode* head, int left, int right) {
    if (!head || left == right) return head;
    ListNode dummy(0);
    dummy.next = head;
    ListNode* prev = &dummy;
    for (int i = 1; i < left; i++) prev = prev->next;
    ListNode* cur = prev->next;
    for (int i = 0; i < right - left; i++) {
        ListNode* tmp = cur->next;
        cur->next = tmp->next;
        tmp->next = prev->next;
        prev->next = tmp;
    }
    return dummy.next;
}

int main() {
    ListNode* head = new ListNode(1);
    head->next = new ListNode(2);
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(4);
    head->next->next->next->next = new ListNode(5);
    head = reverseBetween(head, 2, 4);
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    return 0;
}
```



```
int main() {
    cout << "Original list: ";
    printList(head);

    head = reverseBetween(head, 2, 4);

    cout << "Reversed list: ";
    printList(head);

    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Original list: 1 2 3 4 5
Reversed list: 1 4 3 2 5
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q4. Detect a Cycle in a Linked List

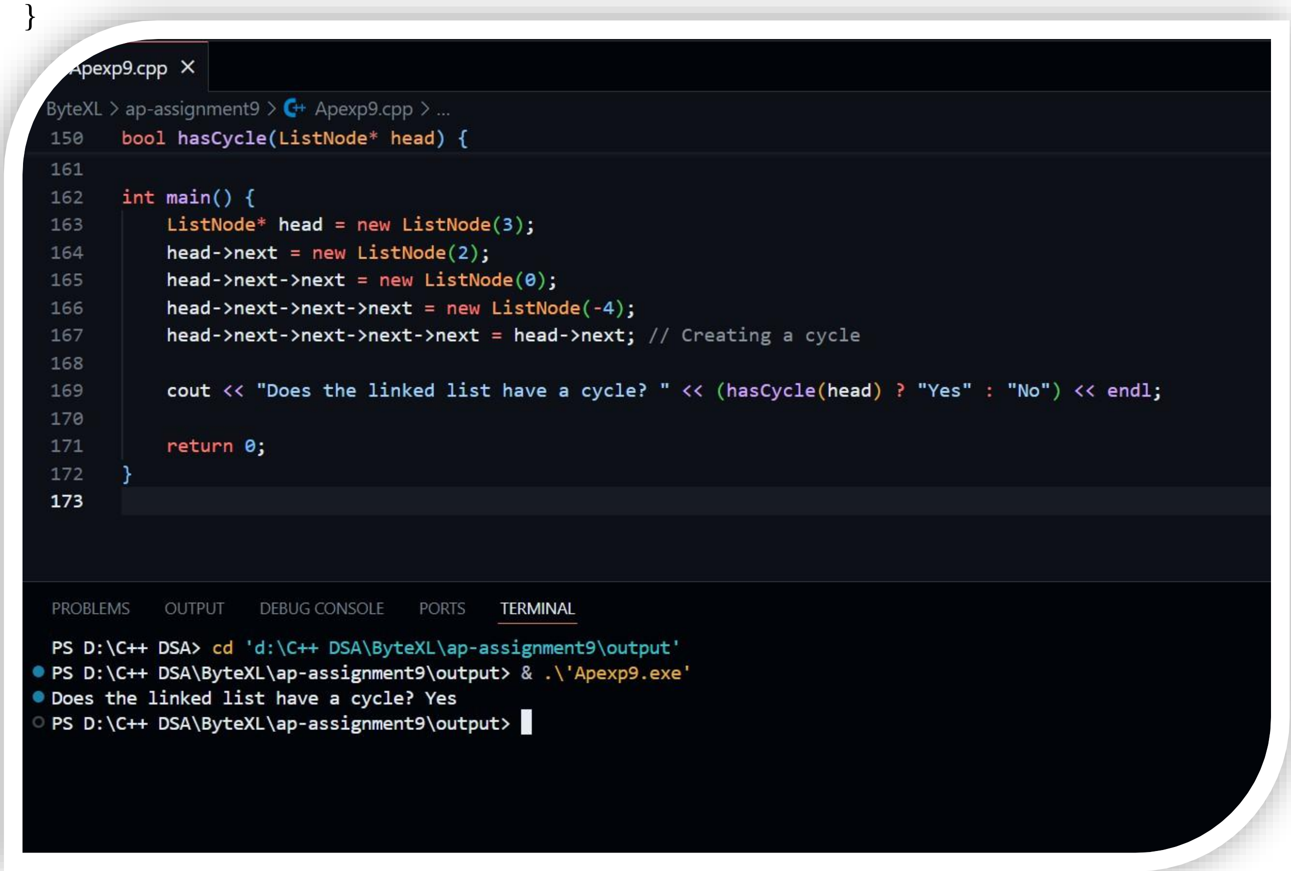
Problem:
Given the head of a linked list, determine whether the linked list contains a cycle.

Approach:
Use Floyd’s Tortoise and Hare algorithm.

Time Complexity: O(n)
Space Complexity: O(1)

Solution:

```
#include <iostream>

using namespace std;


struct ListNode {

    int val;

    ListNode *next;

    ListNode(int x) : val(x), next(nullptr) {}

};


bool hasCycle(ListNode *head) {

    if (!head || !head->next) return false;

    ListNode *slow = head, *fast = head->next;

    while (fast && fast->next) {

        if (slow == fast) return true;

        slow = slow->next;

        fast = fast->next->next;

    }

    return false;

}


int main() {
    ListNode* head = new ListNode(3);

    head->next = new ListNode(2);

    head->next->next = new ListNode(0);

    head->next->next->next = new ListNode(-4);

    head->next->next->next->next = head->next; // cycle

    cout << (hasCycle(head) ? "Cycle Detected" : "No Cycle") << endl;

    return 0;

}
```



```
bool hasCycle(ListNode* head) {

int main() {
    ListNode* head = new ListNode(3);
    head->next = new ListNode(2);
    head->next->next = new ListNode(0);
    head->next->next->next = new ListNode(-4);
    head->next->next->next->next = head->next; // Creating a cycle

    cout << "Does the linked list have a cycle? " << (hasCycle(head) ? "Yes" : "No") << endl;

    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Does the linked list have a cycle? Yes
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q 5. The Skyline Problem

Problem:

Given a list of buildings represented as [left, right, height], return the key points of the skyline.

Approach:

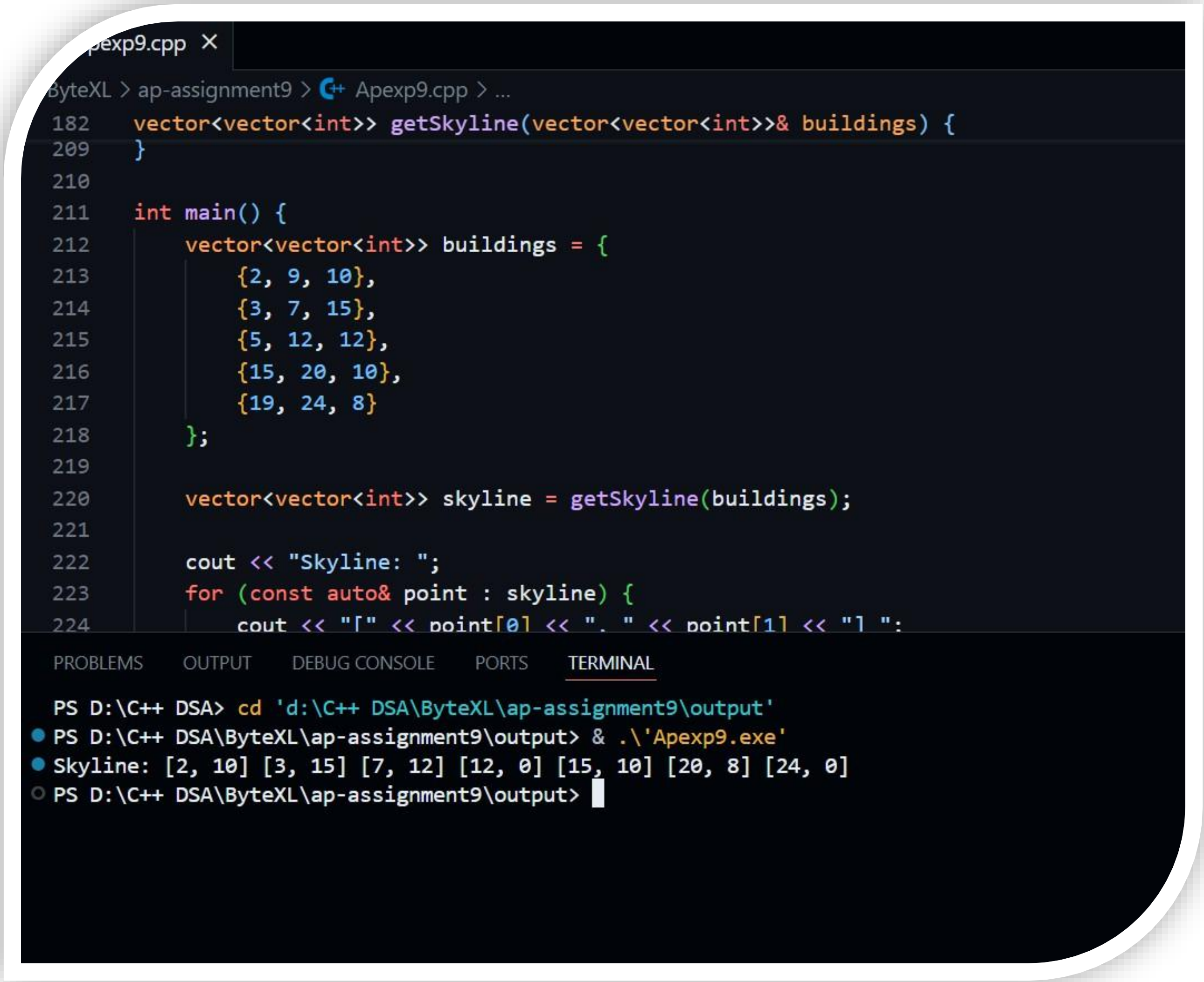
Use sweep line algorithm with max heap.

Time Complexity: O(n log n)
Space Complexity: O(n)

Solution:

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
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};
    vector<vector<int>> result;
    int prevHeight = 0;

    for (const auto& event : events) {
        int x = event.first;
        int h = event.second;
        if (h < 0) {
            heights.insert(-h);
        } else {
            heights.erase(heights.find(h));
        }
        int currentHeight = *heights.rbegin();
        if (currentHeight != prevHeight) {
            result.push_back({x, currentHeight});
            prevHeight = currentHeight;
        }
    }
    return result;
}

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

    return 0;
```

}



```cpp
vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
}

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: ";
    for (const auto& point : skyline) {
        cout << "[" << point[0] << ", " << point[1] << "] ";
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Skyline: [2, 10] [3, 15] [7, 12] [12, 0] [15, 10] [20, 8] [24, 0]
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q 6. Longest Increasing Subsequence II

Problem:

Given an integer array nums, find the length of the longest strictly increasing subsequence.

Approach:

Use patience sorting method with binary search.

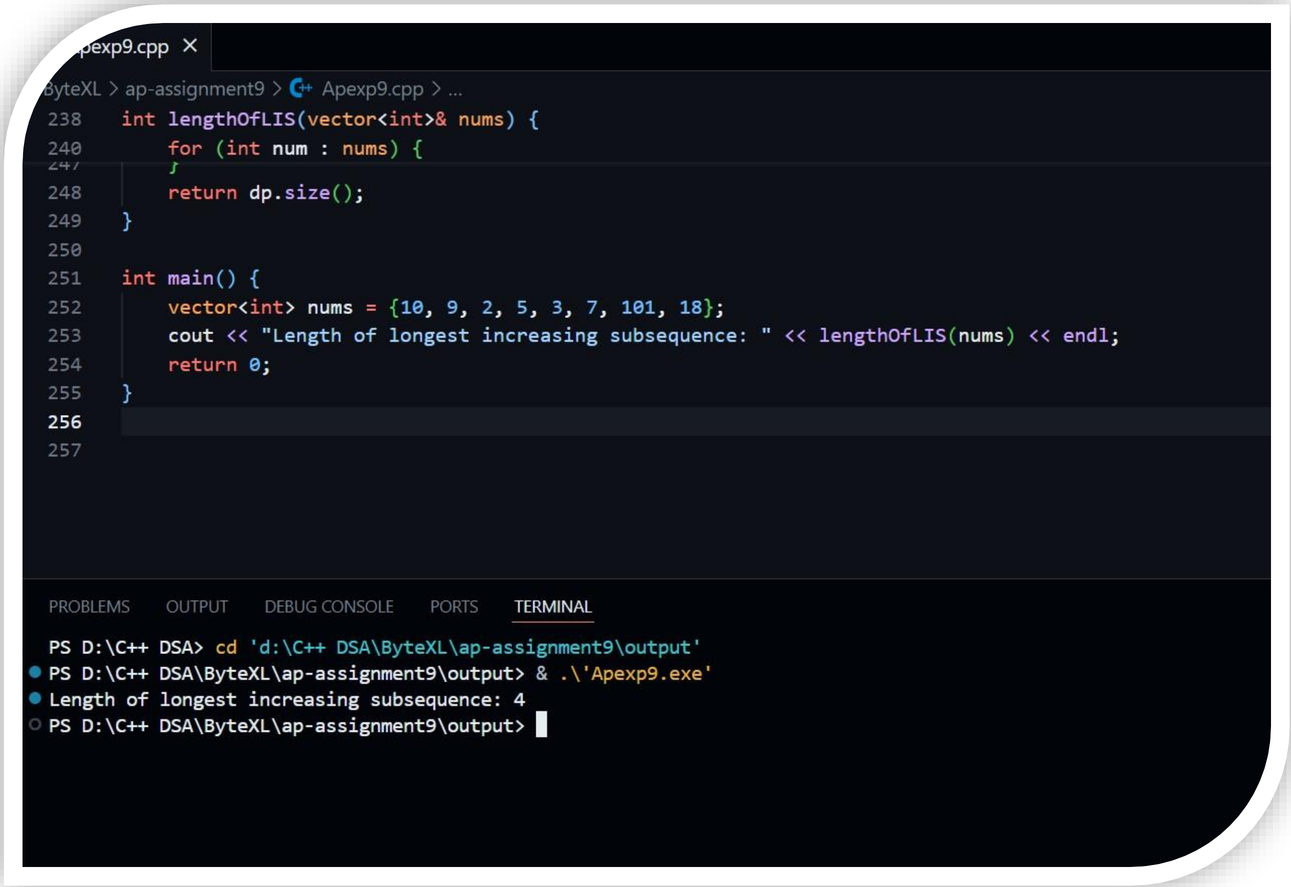
Time Complexity: O(n log n)
Space Complexity: O(n)

Solution:

#include <iostream>

#include <vector>

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

int lengthOfLIS(vector<int>& nums) {
    vector<int> dp;
    for (int x : nums) {
        auto it = lower_bound(dp.begin(), dp.end(), x);
        if (it == dp.end()) dp.push_back(x);
        else *it = x;
    }
    return dp.size();
}

int main() {
    vector<int> nums = {10,9,2,5,3,7,101,18};
    cout << lengthOfLIS(nums) << endl;
    return 0;
}
```



```
int lengthOfLIS(vector<int>& nums) {
    for (int num : nums) {
    }
    return dp.size();
}

int main() {
    vector<int> nums = {10, 9, 2, 5, 3, 7, 101, 18};
    cout << "Length of longest increasing subsequence: " << lengthOfLIS(nums) << endl;
    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Length of longest increasing subsequence: 4
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q7. Search a 2D Matrix II

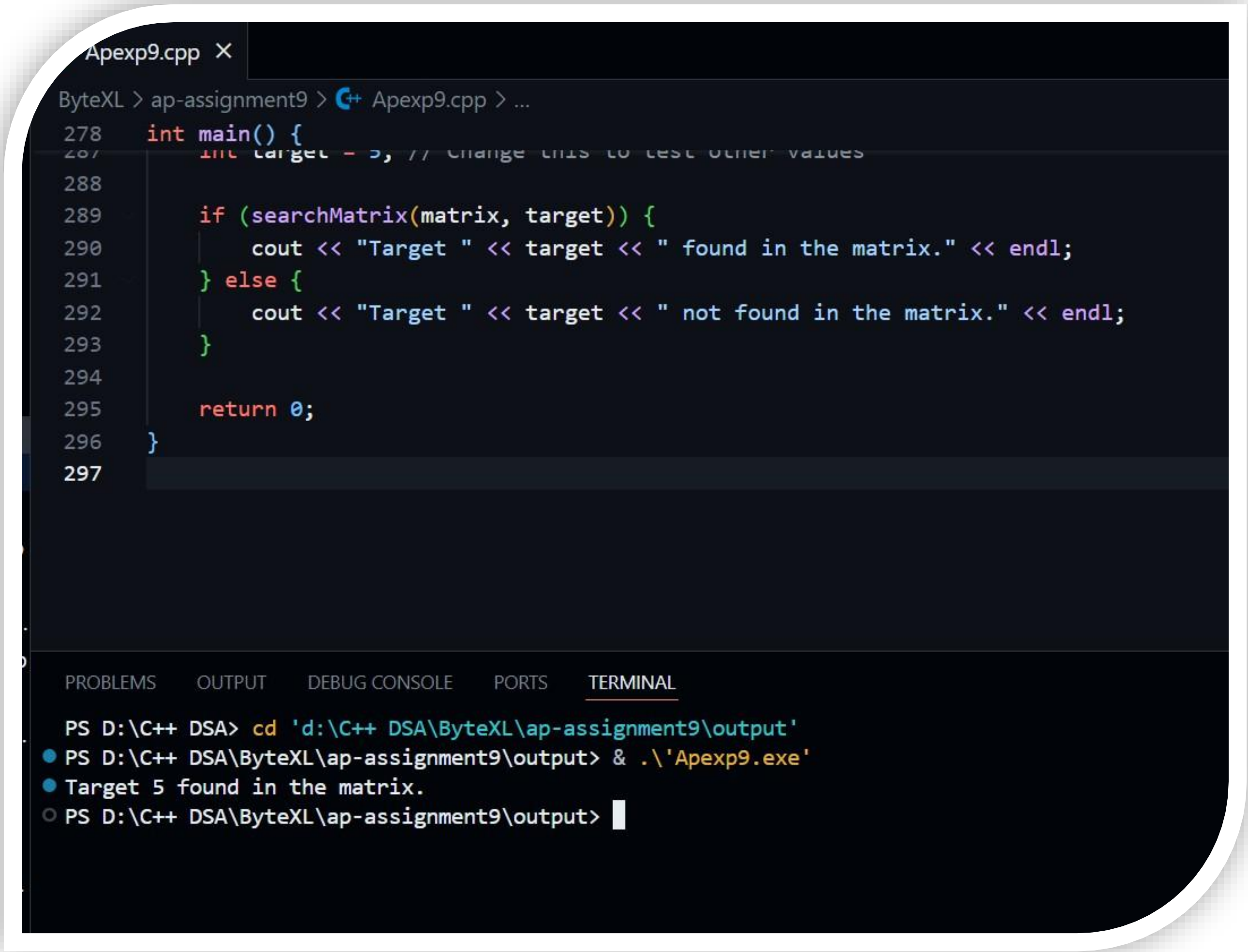
Problem:
Given an m x n matrix where each row is sorted and each column is sorted, determine if target exists in matrix.

Approach:
Start from top-right, move left or down.

Time Complexity: O(m + n)
Space Complexity: O(1)

Solution:

```
#include <iostream>

#include <vector>

using namespace std;


bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size(), n = matrix[0].size();
    int row = 0, col = n - 1;
    while (row < m && col >= 0) {
        if (matrix[row][col] == target) return true;
        else if (matrix[row][col] < target) row++;
        else col--;
    }
    return false;
}

int main() {
    vector<vector<int>> matrix = {{1,4,7,11},{2,5,8,12},{3,6,9,16},{10,13,14,17}};
    int target = 5;
    cout << (searchMatrix(matrix, target) ? "Found" : "Not Found") << endl;
    return 0;
}
```



```
int main() {
    int target = 5; // Change this to test other values

    if (searchMatrix(matrix, target)) {
        cout << "Target " << target << " found in the matrix." << endl;
    } else {
        cout << "Target " << target << " not found in the matrix." << endl;
    }

    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Target 5 found in the matrix.
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q8. Word Break

Problem:
Given a string and a dictionary of words, determine if the string can be segmented into dictionary words.

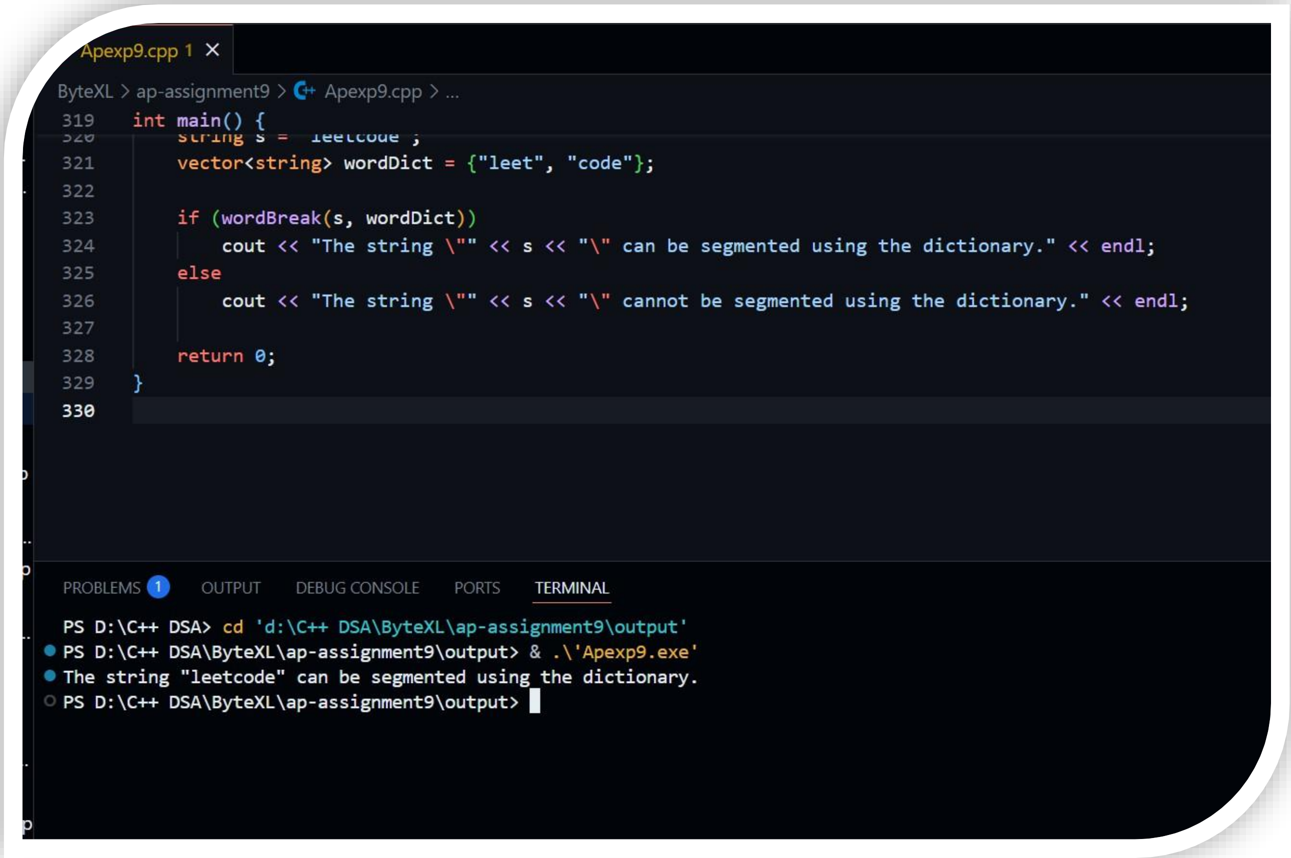
Approach:
Use dynamic programming.

Time Complexity: O(n^2)
Space Complexity: O(n)

Solution:

```
#include <iostream>

#include <vector>

#include <unordered_set>

using namespace std;
```

```
bool wordBreak(string s, vector<string>& wordDict) {
    unordered_set<string> dict(wordDict.begin(), wordDict.end());
    vector<bool> dp(s.size()+1, false);
    dp[0] = true;
    for (int i = 1; i <= s.size(); i++) {
        for (int j = 0; j < i; j++) {
            if (dp[j] && dict.count(s.substr(j, i-j))) {
                dp[i] = true;
                break;
            }
        }
    }
    return dp[s.size()];
}

int main() {
    string s = "leetcode";
    vector<string> dict = {"leet", "code"};
    cout << (wordBreak(s, dict) ? "Yes" : "No") << endl;
    return 0;
}
```



```cpp
int main() {
    string s = "leetcode";
    vector<string> wordDict = {"leet", "code"};

    if (wordBreak(s, wordDict))
        cout << "The string \"" << s << "\" can be segmented using the dictionary." << endl;
    else
        cout << "The string \"" << s << "\" cannot be segmented using the dictionary." << endl;

    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
The string "leetcode" can be segmented using the dictionary.
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q9. Longest Increasing Path in a Matrix

Problem:
Given an m x n integer matrix, find the longest increasing path.

Approach:
Use DFS + memoization.

Time Complexity: O(m * n)
Space Complexity: O(m * n)

Solution:

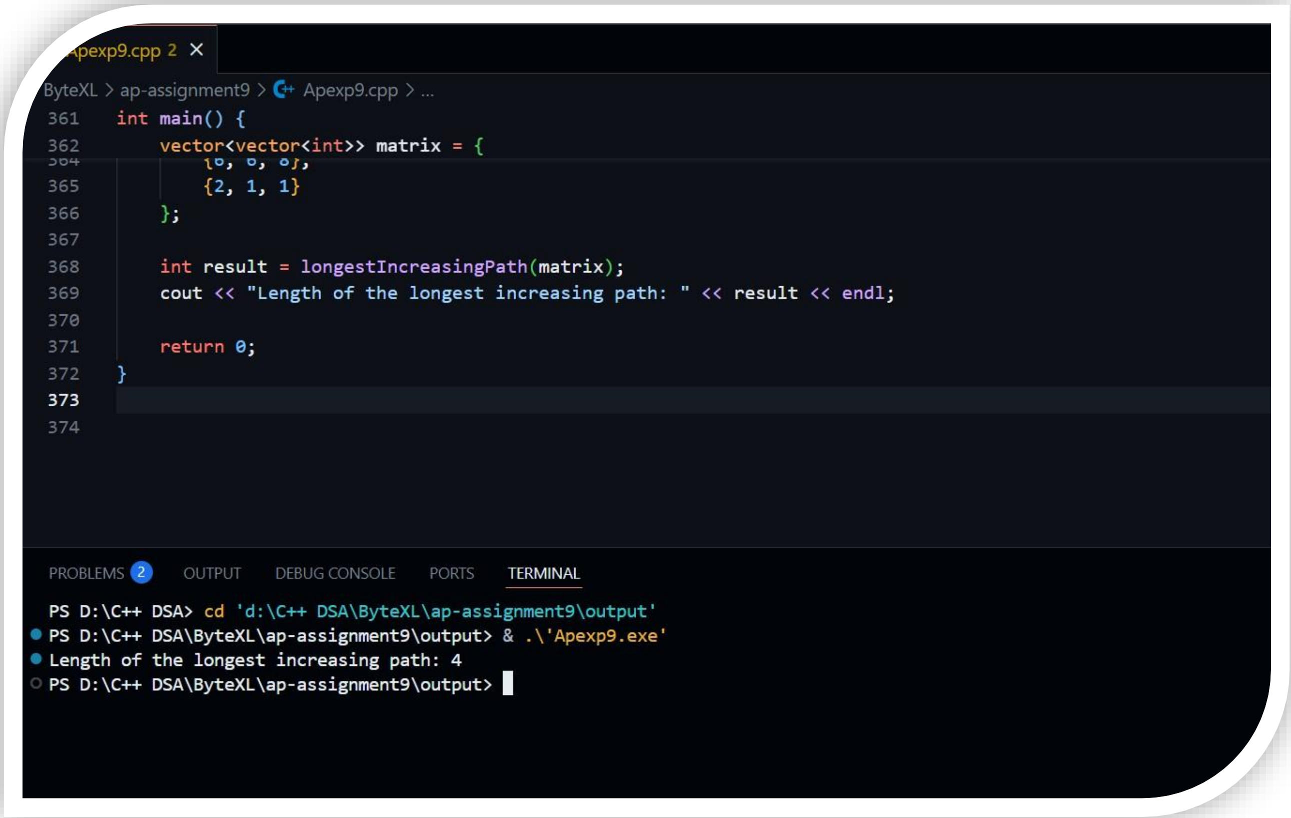
```cpp
#include <iostream>

#include <vector>

using namespace std;


vector<vector<int>> dirs = {{0,1},{1,0},{0,-1},{-1,0}};
```

```
int dfs(vector<vector<int>>& mat, int i, int j, vector<vector<int>>& memo) {
    if (memo[i][j]) return memo[i][j];
    int maxLen = 1;
    for (auto& d : dirs) {
        int x = i + d[0], y = j + d[1];
        if (x >= 0 && x < mat.size() && y >= 0 && y < mat[0].size() && mat[x][y] > mat[i][j]) {
            maxLen = max(maxLen, 1 + dfs(mat, x, y, memo));
        }
    }
    return memo[i][j] = maxLen;
}

int longestIncreasingPath(vector<vector<int>>& matrix) {
    if (matrix.empty()) return 0;
    int m = matrix.size(), n = matrix[0].size();
    vector<vector<int>> memo(m, vector<int>(n, 0));
    int res = 0;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            res = max(res, dfs(matrix, i, j, memo));
        }
    }
    return res;
}

int main() {
    vector<vector<int>> mat = {{9,9,4},{6,6,8},{2,1,1}};
```

    cout << longestIncreasingPath(mat) << endl;

    return 0;

}



```
int main() {
    vector<vector<int>> matrix = {
        {6, 6, 8},
        {2, 1, 1}
    };

    int result = longestIncreasingPath(matrix);
    cout << "Length of the longest increasing path: " << result << endl;

    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Length of the longest increasing path: 4
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```

Q10. Trapping Rain Water

Problem:
Given n non-negative integers representing elevation map, compute how much water can be trapped.

Approach:
Use two-pointer technique.

Time Complexity: O(n)
Space Complexity: O(1)

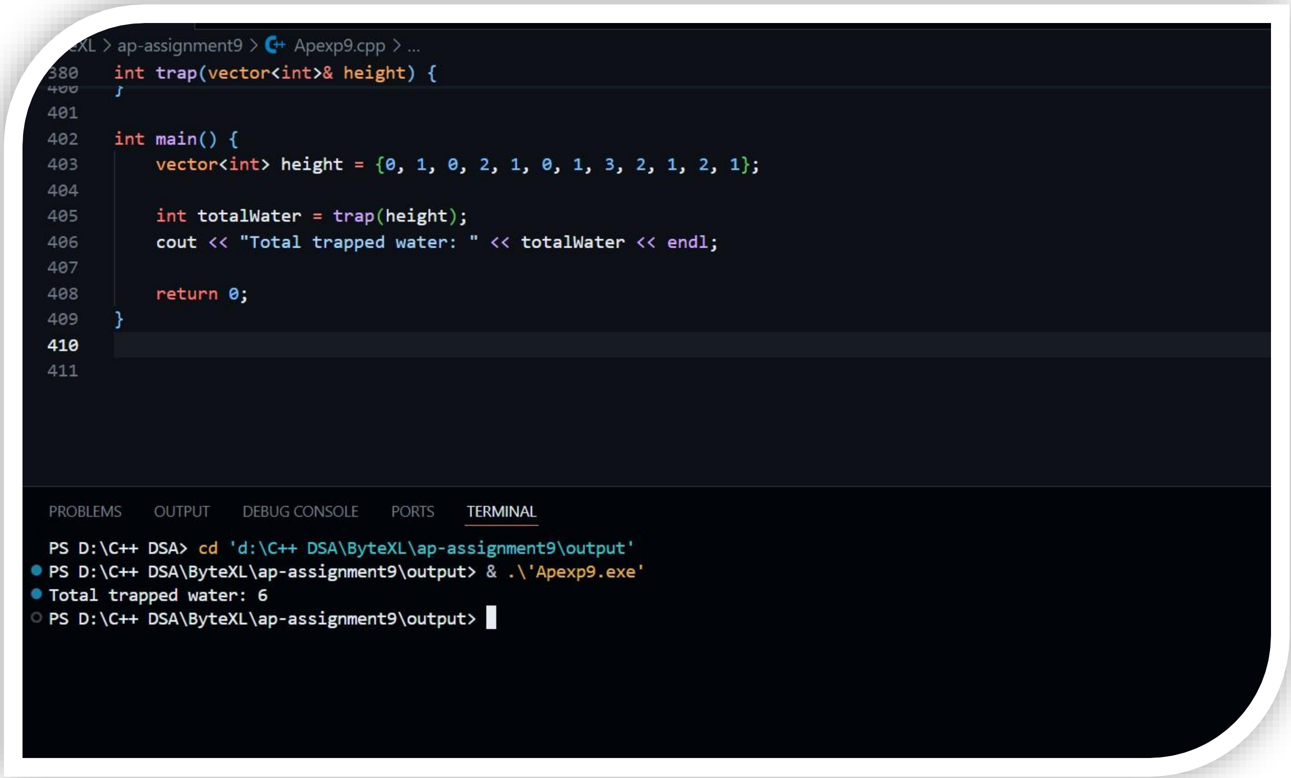
Solution:

#include <iostream>

#include <vector>

using namespace std;

```
int trap(vector<int>& height) {
    int left = 0, right = height.size() - 1, leftMax = 0, rightMax = 0, res = 0;
    while (left < right) {
        if (height[left] < height[right]) {
            if (height[left] >= leftMax) leftMax = height[left];
            else res += leftMax - height[left];
            left++;
        } else {
            if (height[right] >= rightMax) rightMax = height[right];
            else res += rightMax - height[right];
            right--;
        }
    }
    return res;
}

int main() {
    vector<int> height = {0,1,0,2,1,0,1,3,2,1,2,1};
    cout << trap(height) << endl;
    return 0;
}
```



```cpp
int trap(vector<int>& height) {
}

int main() {
    vector<int> height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};

    int totalWater = trap(height);
    cout << "Total trapped water: " << totalWater << endl;

    return 0;
}
```

```
PS D:\C++ DSA> cd 'd:\C++ DSA\ByteXL\ap-assignment9\output'
PS D:\C++ DSA\ByteXL\ap-assignment9\output> & .\'Apexp9.exe'
Total trapped water: 6
PS D:\C++ DSA\ByteXL\ap-assignment9\output>
```