Name: Vivek Tiwari

Email: [vivek120303@gmail.com](mailto:vivek120303@gmail.com)

Contact: 9022841987

Question 1:  
Write a function to reverse a singly linked list. The function should take the head of the list and return the new head of the reversed list.

Code:

#include <bits/stdc++.h>

using namespace std;

// Definition for singly-linked list.

struct ListNode {

    int val;

    ListNode \*next;

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

};

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        ListNode\* prev = nullptr;

        ListNode\* current = head;

        ListNode\* next = nullptr;

        while (current != nullptr) {

            next = current->next;  // store next node

            current->next = prev;  // reverse the current node's pointer

            prev = current;        // move pointers one position ahead

            current = next;

        }

        return prev;

    }

};

// Helper function to print the list

void printList(ListNode\* head) {

    ListNode\* current = head;

    while (current != nullptr) {

        std::cout << current->val << " ";

        current = current->next;

    }

    std::cout << std::endl;

}

int main() {

    // Creating a sample list: 1 -> 2 -> 3 -> 4 -> 5

    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);

    std::cout << "Original list: ";

    printList(head);

    Solution solution;

    ListNode\* reversedHead = solution.reverseList(head);

    std::cout << "Reversed list: ";

    printList(reversedHead);

    // Clean up the memory

    while (reversedHead != nullptr) {

        ListNode\* temp = reversedHead;

        reversedHead = reversedHead->next;

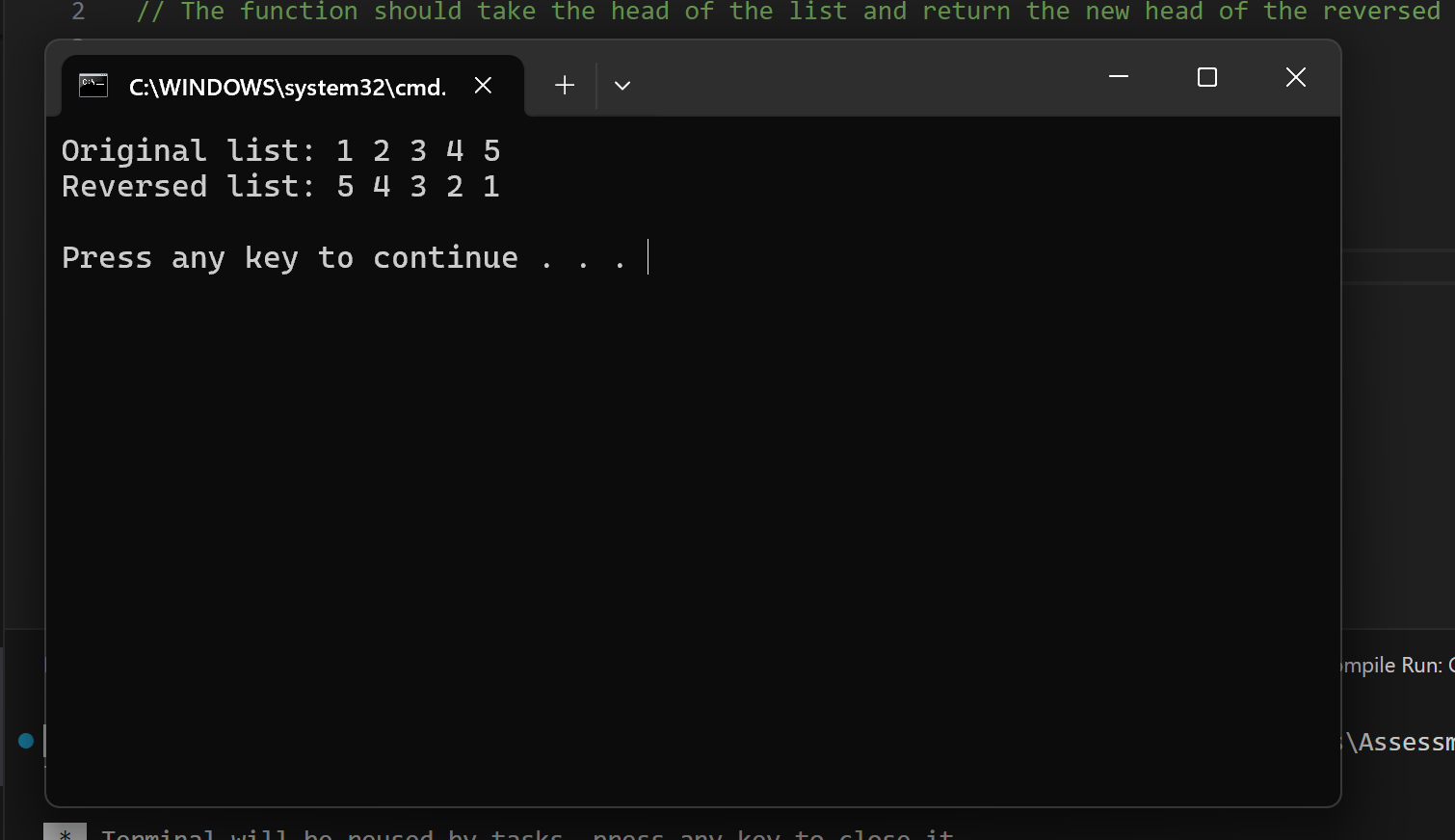
        delete temp;

    }

    return 0;

}

Output:



Question 2:  
Given a string, find the length of the longest substring without repeating characters.The function should return an integer representing the length of the longest substring without repeating characters.

Code:

#include <iostream>

#include <unordered\_map>

#include <string>

class Solution {

public:

    int lengthOfLongestSubstring(const std::string& s) {

        std::unordered\_map<char, int> charIndexMap;

        int maxLength = 0, start = 0;

        for (int end = 0; end < s.size(); ++end) {

            if (charIndexMap.find(s[end]) != charIndexMap.end()) {

                start = std::max(charIndexMap[s[end]] + 1, start);

            }

            charIndexMap[s[end]] = end;

            maxLength = std::max(maxLength, end - start + 1);

        }

        return maxLength;

    }

};

int main() {

    Solution solution;

    std::string s;

    // Example usage

    s = "abcabcbb";

    std::cout << "The length of the longest substring without repeating characters in \"" << s << "\" is: "

              << solution.lengthOfLongestSubstring(s) << std::endl;

    s = "bbbbb";

    std::cout << "The length of the longest substring without repeating characters in \"" << s << "\" is: "

              << solution.lengthOfLongestSubstring(s) << std::endl;

    s = "pwwkew";

    std::cout << "The length of the longest substring without repeating characters in \"" << s << "\" is: "

              << solution.lengthOfLongestSubstring(s) << std::endl;

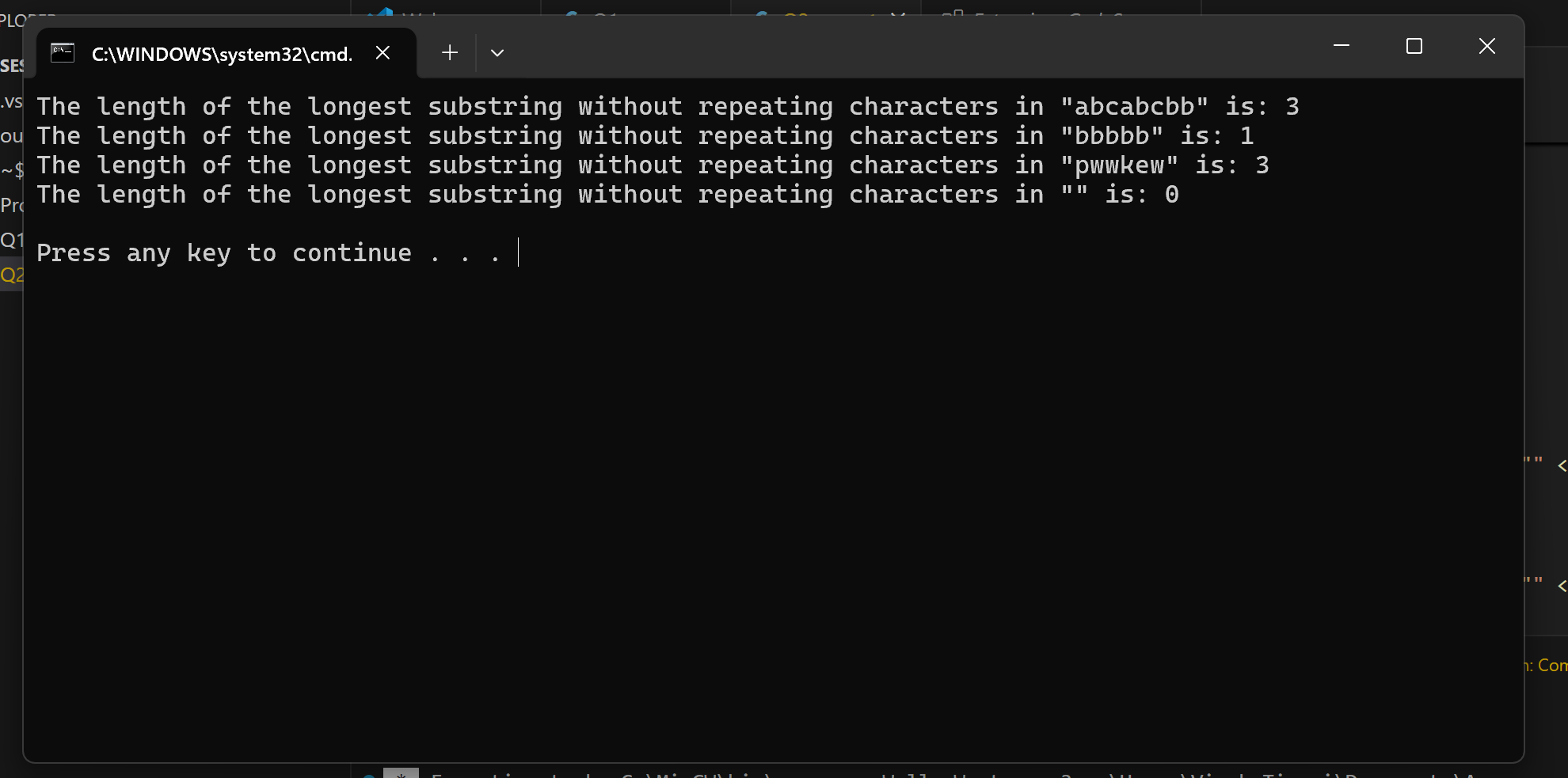
    s = "";

    std::cout << "The length of the longest substring without repeating characters in \"" << s << "\" is: "

              << solution.lengthOfLongestSubstring(s) << std::endl;

    return 0;

}

Output:  


Question 3:  
Given a non-empty binary tree, find the maximum path sum. A path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain at least one node and does not need to go through the root.The function should return an integer representing the maximum path sum.  
  
Code:

#include <iostream>

#include <climits>

// Definition for a binary tree node.

struct TreeNode {

    int val;

    TreeNode \*left;

    TreeNode \*right;

    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

    int maxPathSum(TreeNode\* root) {

        int maxSum = INT\_MIN;

        maxGain(root, maxSum);

        return maxSum;

    }

private:

    int maxGain(TreeNode\* node, int& maxSum) {

        if (node == nullptr) return 0;

        int leftGain = std::max(maxGain(node->left, maxSum), 0);

        int rightGain = std::max(maxGain(node->right, maxSum), 0);

        int currentPathSum = node->val + leftGain + rightGain;

        maxSum = std::max(maxSum, currentPathSum);

        return node->val + std::max(leftGain, rightGain);

    }

};

int main() {

    TreeNode\* root = new TreeNode(-10);

    root->left = new TreeNode(9);

    root->right = new TreeNode(20);

    root->right->left = new TreeNode(15);

    root->right->right = new TreeNode(7);

    // Tree is

    //         -10

    //        /   \

    //       9     20

    //             / \

    //            15  7

    Solution solution;

    std::cout << "Maximum path sum is: " << solution.maxPathSum(root) << std::endl;

    // Cleanup

    delete root->right->right;

    delete root->right->left;

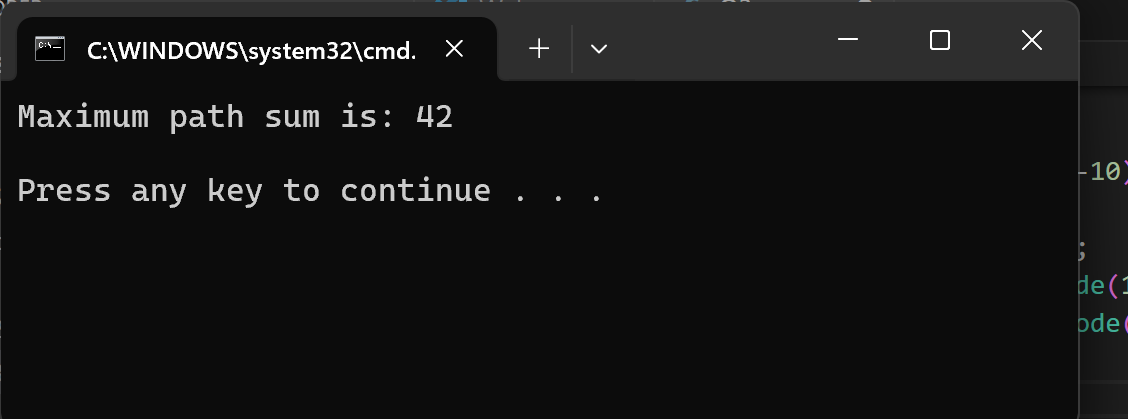
    delete root->right;

    delete root->left;

    delete root;

    return 0;

}

Output:  
  
  
  
Question 4:  
Design an algorithm to serialize and deserialize a binary tree. Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment. Implement the serialize and deserialize methods.

Code:

#include <iostream>

#include <sstream>

#include <queue>

#include <string>

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) {}

};

class Codec {

public:

    // Encodes a tree to a single string.

    string serialize(TreeNode\* root) {

        if (!root) return "";

        string data = "";

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            TreeNode\* node = q.front();

            q.pop();

            if (!node) {

                data += "#,";

                continue;

            } else {

                data += to\_string(node->val) + ',';

                q.push(node->left);

                q.push(node->right);

            }

        }

        data.pop\_back();

        return data;

    }

    // Decodes your encoded data to tree.

    TreeNode\* deserialize(string data) {

        if (data.size() == 0) return NULL;

        if (data[0] == '#') return NULL;

        stringstream ss(data);

        string node;

        getline(ss, node, ',');

        TreeNode\* root = new TreeNode(stoi(node));

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            TreeNode\* curr = q.front();

            q.pop();

            getline(ss, node, ',');

            if (node == "#") {

                curr->left = NULL;

            } else {

                TreeNode\* leftNode = new TreeNode(stoi(node));

                curr->left = leftNode;

                q.push(leftNode);

            }

            getline(ss, node, ',');

            if (node == "#") {

                curr->right = NULL;

            } else {

                TreeNode\* rightNode = new TreeNode(stoi(node));

                curr->right = rightNode;

                q.push(rightNode);

            }

        }

        return root;

    }

};

void printTree(TreeNode\* root) {

    if (!root) return;

    queue<TreeNode\*> q;

    q.push(root);

    while (!q.empty()) {

        TreeNode\* node = q.front();

        q.pop();

        if (node) {

            cout << node->val << " ";

            q.push(node->left);

            q.push(node->right);

        } else {

            cout << "# ";

        }

    }

    cout << endl;

}

int main() {

    Codec codec;

    // Create a sample tree

    TreeNode\* root = new TreeNode(1);

    root->left = new TreeNode(2);

    root->right = new TreeNode(3);

    root->right->left = new TreeNode(4);

    root->right->right = new TreeNode(5);

    // Serialize the tree

    string serialized = codec.serialize(root);

    cout << "Serialized tree: " << serialized << endl;

    // Deserialize the string back to a tree

    TreeNode\* deserializedRoot = codec.deserialize(serialized);

    // Print the deserialized tree

    cout << "Deserialized tree: ";

    printTree(deserializedRoot);

    // Cleanup

    delete root->right->right;

    delete root->right->left;

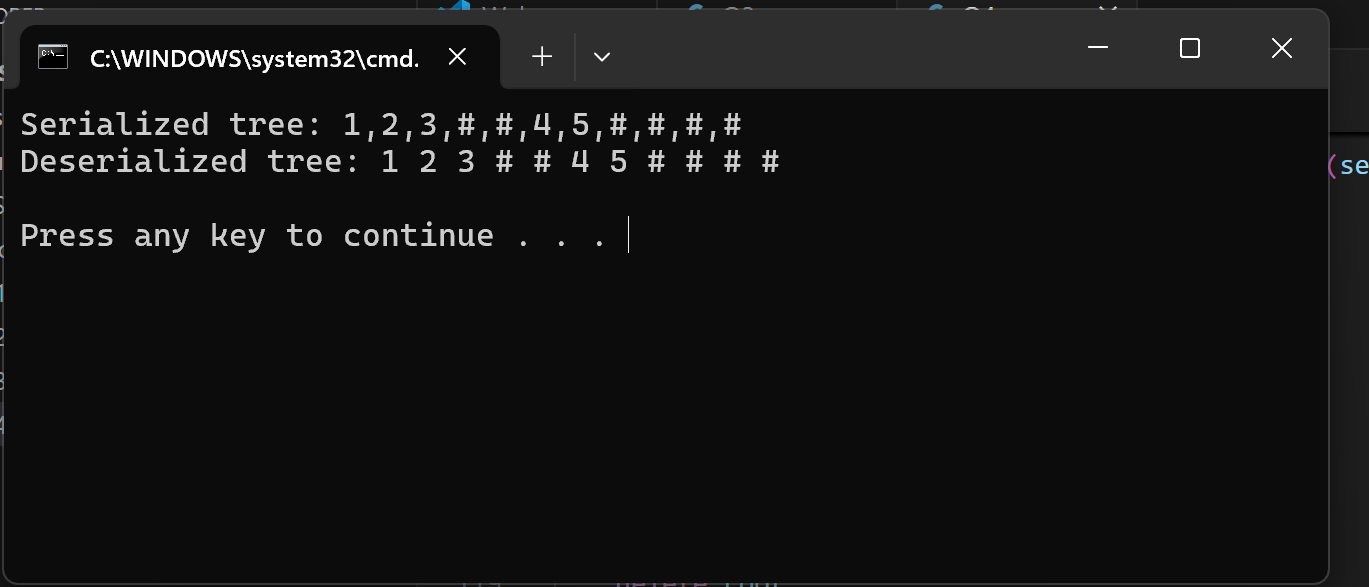
    delete root->right;

    delete root->left;

    delete root;

    return 0;

}

Output:  


Question 5:  
Write a function to rotate an array to the right by k steps. The function should modify the array in place to achieve the rotation.  
  
Code:

#include <iostream>

#include <vector>

#include <algorithm>

// Function to rotate the array to the right by k steps

void rotate(std::vector<int>& nums, int k) {

    int n = nums.size();

    k %= n; // If k is greater than n, take the modulus

    // Reverse the whole array

    std::reverse(nums.begin(), nums.end());

    // Reverse the first k elements

    std::reverse(nums.begin(), nums.begin() + k);

    // Reverse the remaining elements

    std::reverse(nums.begin() + k, nums.end());

}

void printArray(const std::vector<int>& nums) {

    for (int num : nums) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

}

int main() {

    std::vector<int> nums = {1, 2, 3, 4, 5, 6, 7};

    int k = 3;

    std::cout << "Original array: ";

    printArray(nums);

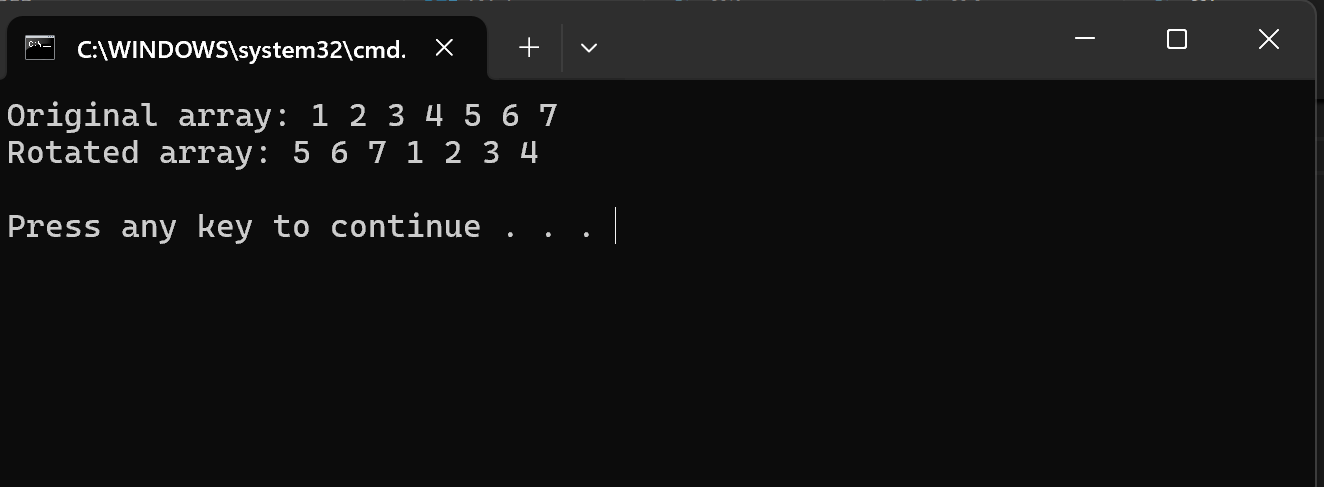
    rotate(nums, k);

    std::cout << "Rotated array: ";

    printArray(nums);

    return 0;

}

Output:  


Question 6:  
Write a function to find the factorial of a given number. The function should return the factorial of the number.

Code:

#include <iostream>

// Function to calculate the factorial of a given number

unsigned long long factorial(int n) {

    // Handling the base case for 0 and 1

    if (n == 0 || n == 1) return 1;

    unsigned long long result = 1;

    for (int i = 2; i <= n; ++i) {

        result \*= i;

    }

    return result;

}

int main() {

    int number;

    std::cout << "Enter a number to find its factorial: ";

    std::cin >> number;

    // Validate input

    if (number < 0) {

        std::cout << "Factorial is not defined for negative numbers." << std::endl;

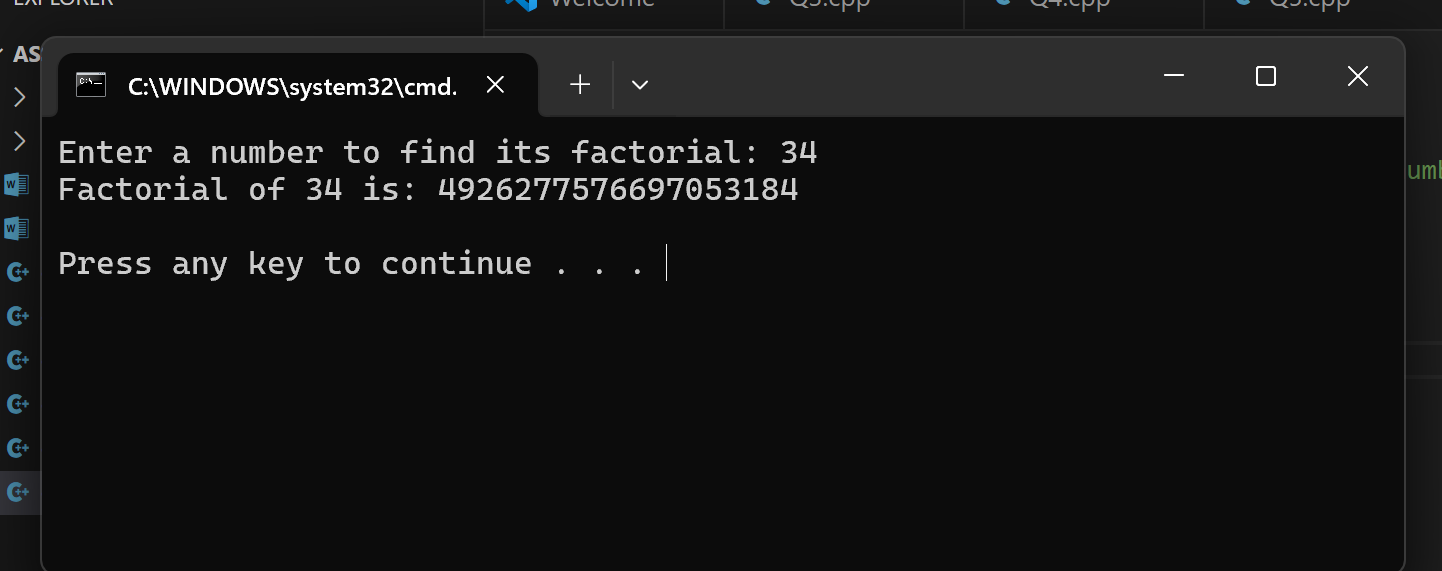
    } else {

        std::cout << "Factorial of " << number << " is: " << factorial(number) << std::endl;

    }

    return 0;

}

Output:  


Question 7:  
Write a function to compute the sum of the digits of a given number .The function should return the sum of the digits of the number.  
  
Code:

#include <iostream>

// Function to compute the sum of the digits of a given number

int sumOfDigits(int num) {

    int sum = 0;

    num = abs(num); // Handle negative numbers by taking their absolute value

    while (num > 0) {

        sum += num % 10; // Extract the last digit

        num /= 10;       // Remove the last digit

    }

    return sum;

}

int main() {

    int number;

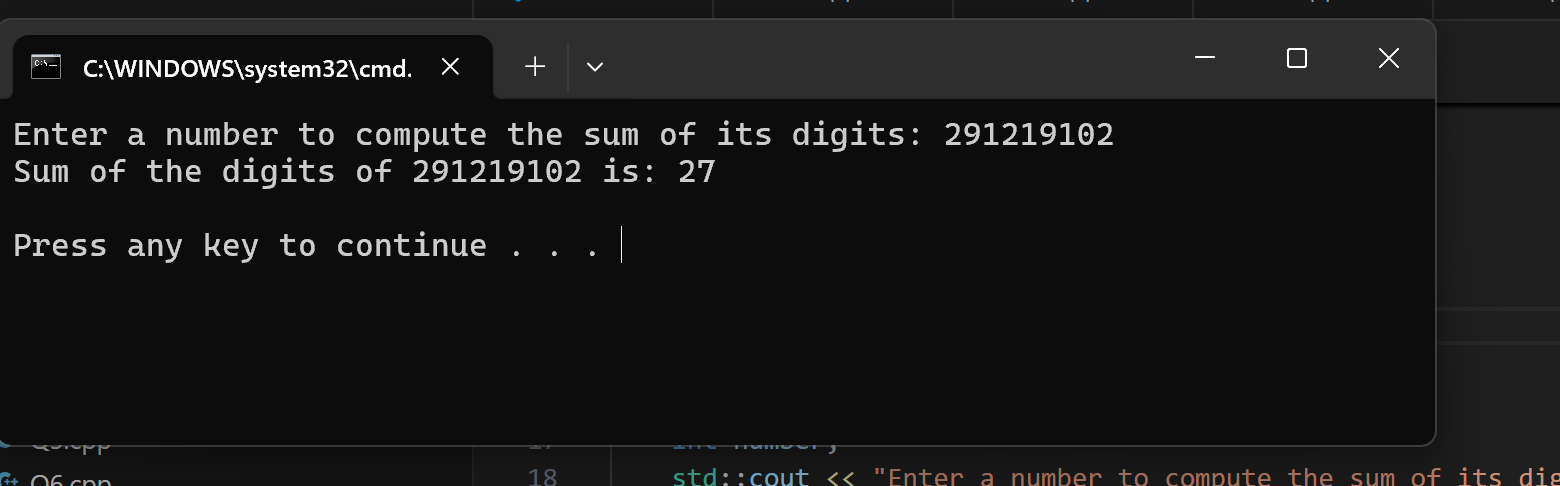
    std::cout << "Enter a number to compute the sum of its digits: ";

    std::cin >> number;

    std::cout << "Sum of the digits of " << number << " is: " << sumOfDigits(number) << std::endl;

    return 0;

}

Output:  


Question 8:  
Write a function to find the greatest common divisor (GCD) of two numbers. The function should return the GCD of a and b.  
  
Code;

#include <bits/stdc++.h>

using namespace std;

// Function to find the GCD of two numbers using the Euclidean algorithm

int gcd(int a, int b) {

    while (b != 0) {

        int temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

int main() {

    int num1, num2;

    std::cout << "Enter two numbers to find their GCD: ";

    std::cin >> num1 >> num2;

    cout<<"Using Euclidean algorithm"<<endl;

    std::cout << "The GCD of " << num1 << " and " << num2 << " is: " << gcd(num1, num2) << std::endl;

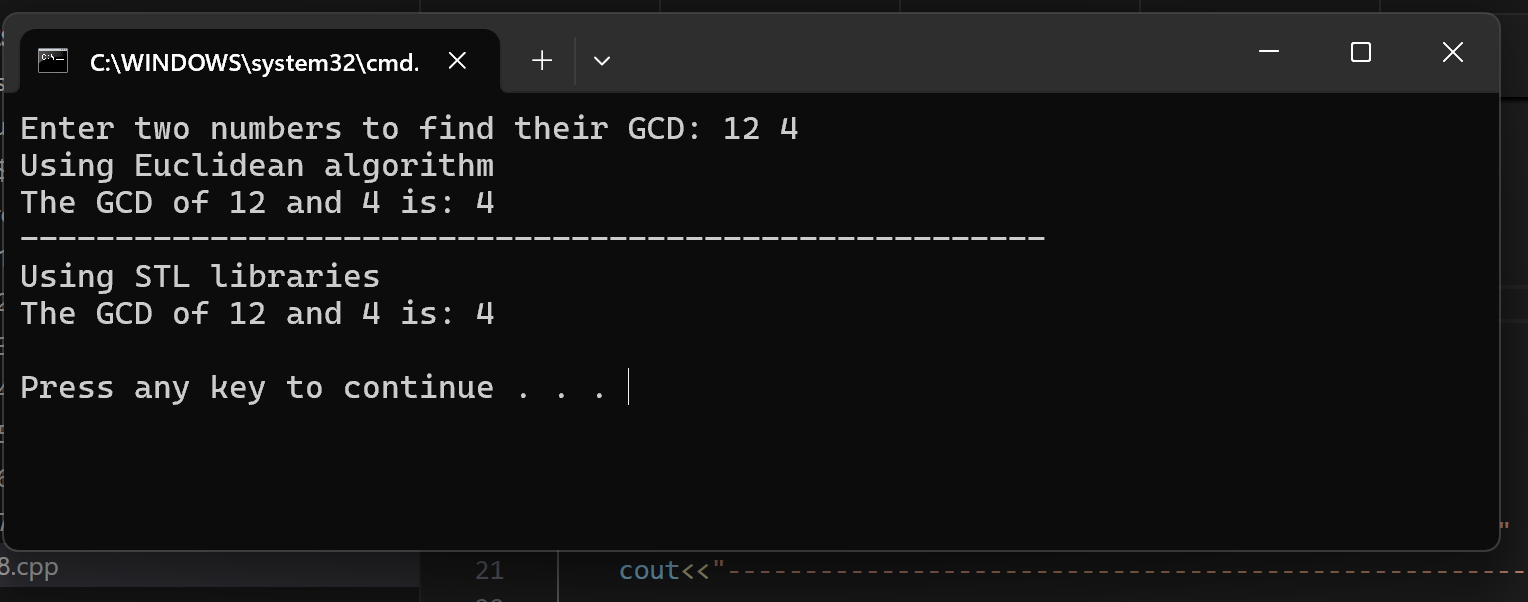
    cout<<"------------------------------------------------------\n";

    cout<<"Using STL libraries"<<endl;

    std::cout << "The GCD of " << num1 << " and " << num2 << " is: " << \_\_gcd(num1, num2) << std::endl;

    return 0;

}

Output:  


Question 9:  
Write a function to find the maximum difference between any two elements in an array.The function should return the maximum difference between any two elements in the array.

Code:

#include <bits/stdc++.h>

using namespace std;

// Function to find the maximum difference between any two elements in the array

int maxDifference(const std::vector<int>& arr) {

    if (arr.size() < 2) {

        std::cerr << "Array must contain at least two elements." << std::endl;

        return -1;

    }

    int minElement=INT\_MAX, maxElement=INT\_MIN;

    for(auto it:arr){

        minElement=min(minElement,it);

        maxElement=max(maxElement,it);

    }

    return maxElement-minElement;

}

int main() {

    std::vector<int> arr = {7, 1, 5, 3, 6, 4};

    int result = maxDifference(arr);

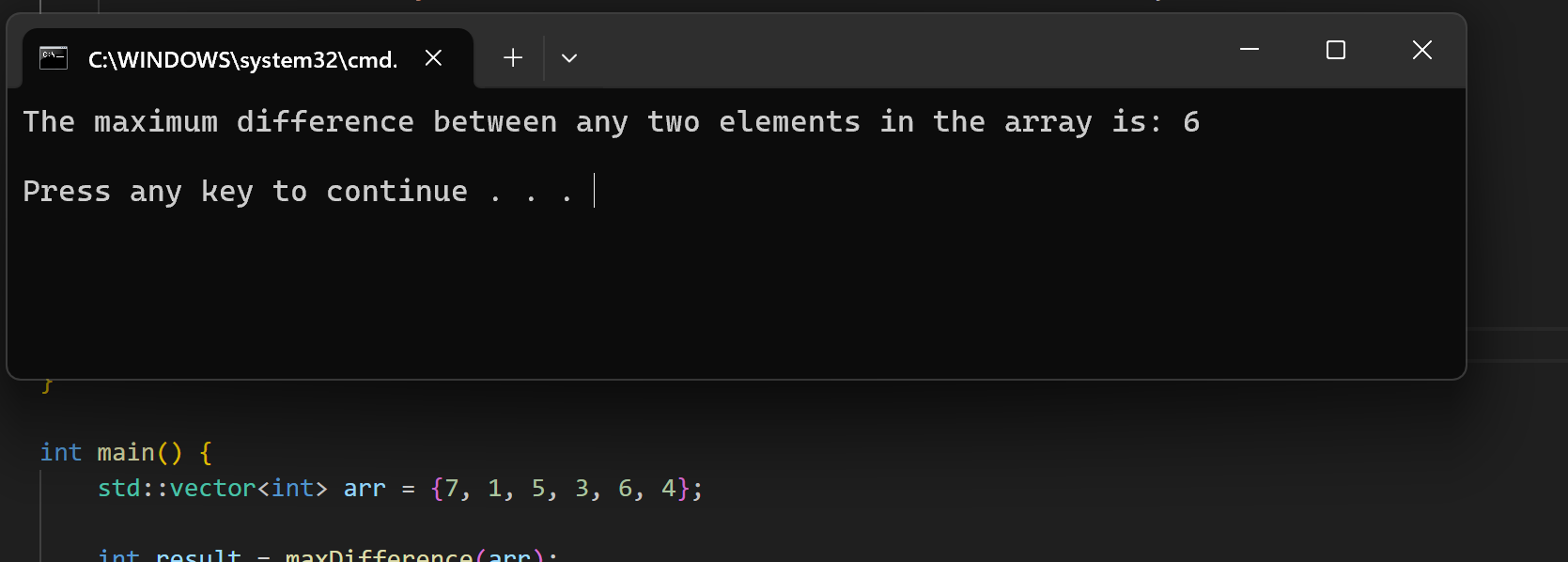
    if (result != -1) {

        std::cout << "The maximum difference between any two elements in the array is: " << result << std::endl;

    }

    return 0;

}

Output:  


Question 10:

Code:

#include <bits/stdc++.h>

using namespace std;

// Function to check if a string contains only alphabetic characters

bool containsOnlyAlphabets(const std::string& str) {

    for (char c : str) {

        if (!std::isalpha(c)) {

            return false;

        }

    }

    return true;

}

int main() {

    std::string str1 = "HelloWorld";

    std::string str2 = "Hello123";

    // Check if str1 contains only alphabetic characters

    if(containsOnlyAlphabets(str1)) {

        std::cout << "\"" << str1 << "\" contains only alphabetic characters." << std::endl;

    }

    else{

        std::cout << "\"" << str1 << "\" does not contain only alphabetic characters." << std::endl;

    }

    // Check if str2 contains only alphabetic characters

    if(containsOnlyAlphabets(str2)) {

        std::cout << "\"" << str2 << "\" contains only alphabetic characters." << std::endl;

    }

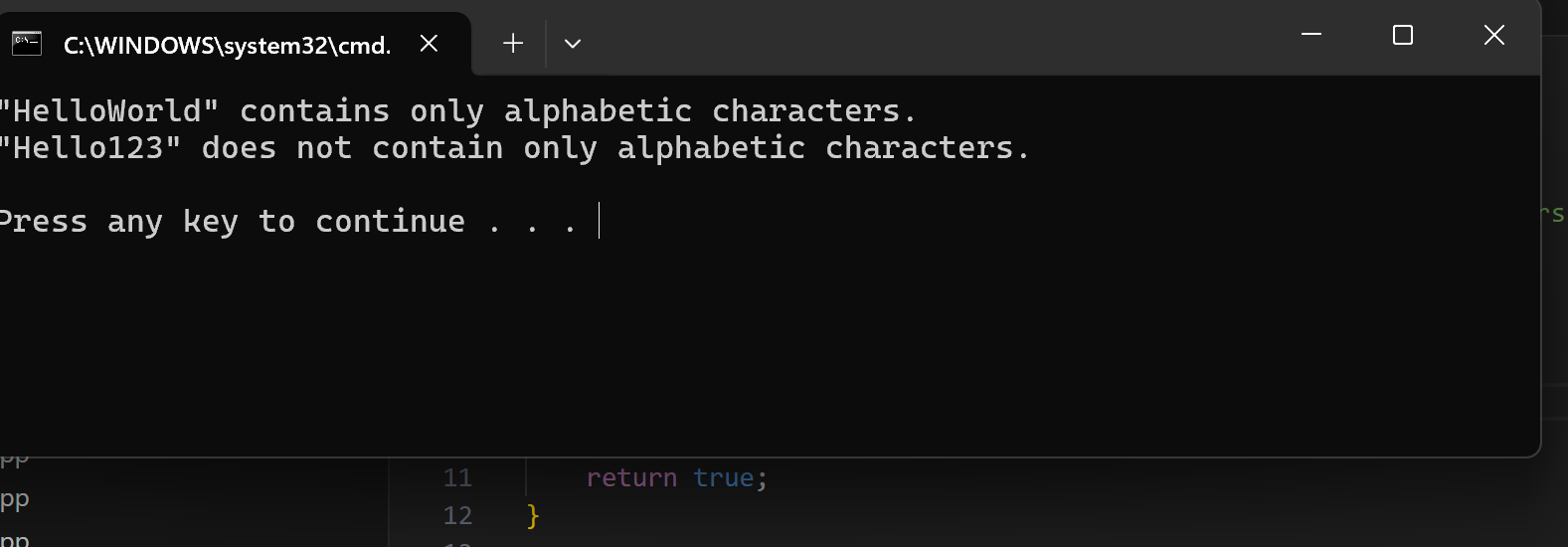
    else{

        std::cout << "\"" << str2 << "\" does not contain only alphabetic characters." << std::endl;

    }

    return 0;

}

Output:  


----------------------------------------------------------- Thank You -----------------------------------------------------------