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.

Solution:

#include <iostream>

struct ListNode {

    int val;

    ListNode\* next;

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

};

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        ListNode\* prev = nullptr;

        ListNode\* curr = head;

        ListNode\* next = nullptr;

        while (curr != nullptr) {

            next = curr->next;

            curr->next = prev;

            prev = curr;

            curr = next;

        }

        return prev;

    }

};

void printList(ListNode\* node) {

    while (node != nullptr) {

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

        node = node->next;

    }

    std::cout << std::endl;

}

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);

    std::cout << "Original list: ";

    printList(head);

    Solution sol;

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

    std::cout << "Reversed list: ";

    printList(reversedHead);

    ListNode\* current = reversedHead;

    while (current != nullptr) {

        ListNode\* next = current->next;

        delete current;

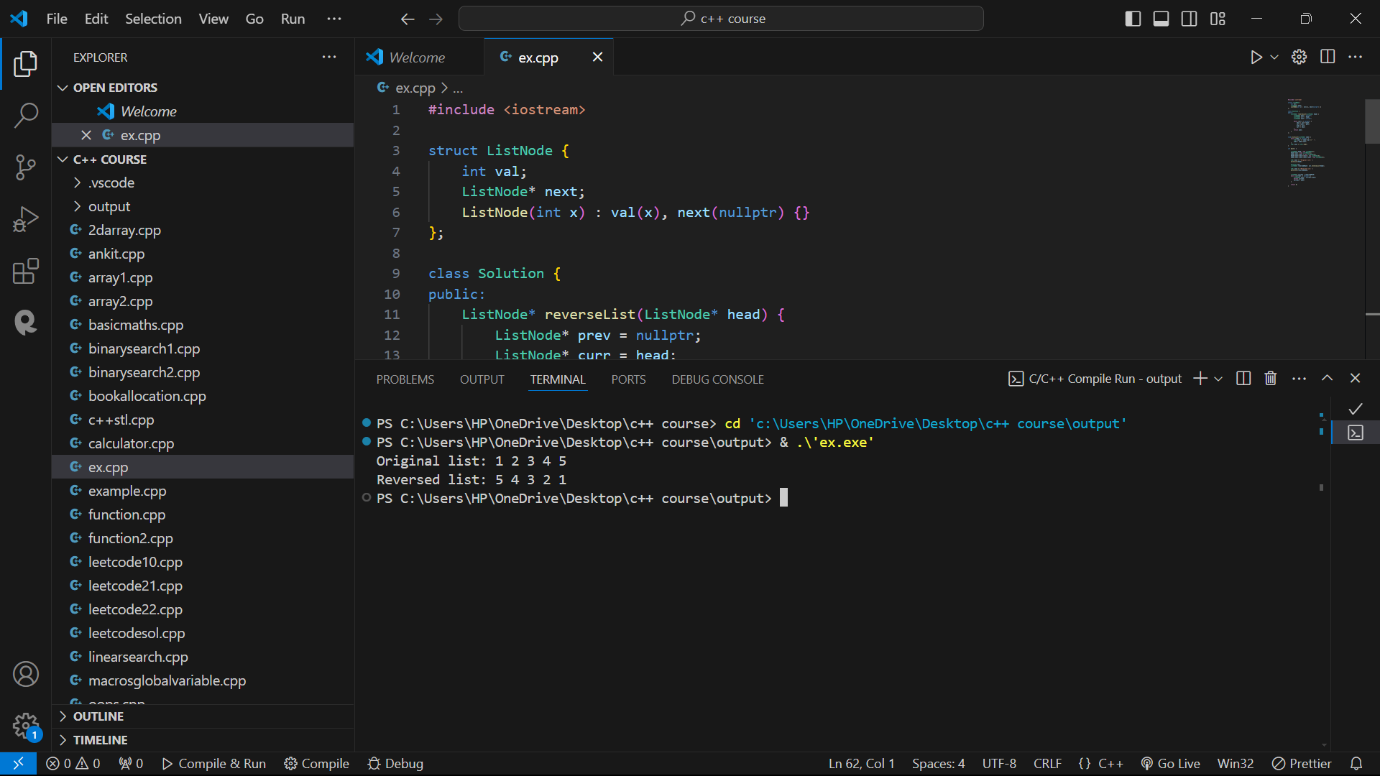
        current = next;

    }

    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.

Solution:

#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 i = 0; i < s.length(); ++i) {

            if (charIndexMap.find(s[i]) != charIndexMap.end() && charIndexMap[s[i]] >= start) {

                start = charIndexMap[s[i]] + 1;

            }

            charIndexMap[s[i]] = i;

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

        }

        return maxLength;

    }

};

int main() {

    Solution sol;

    std::string str = "abcabcbb";

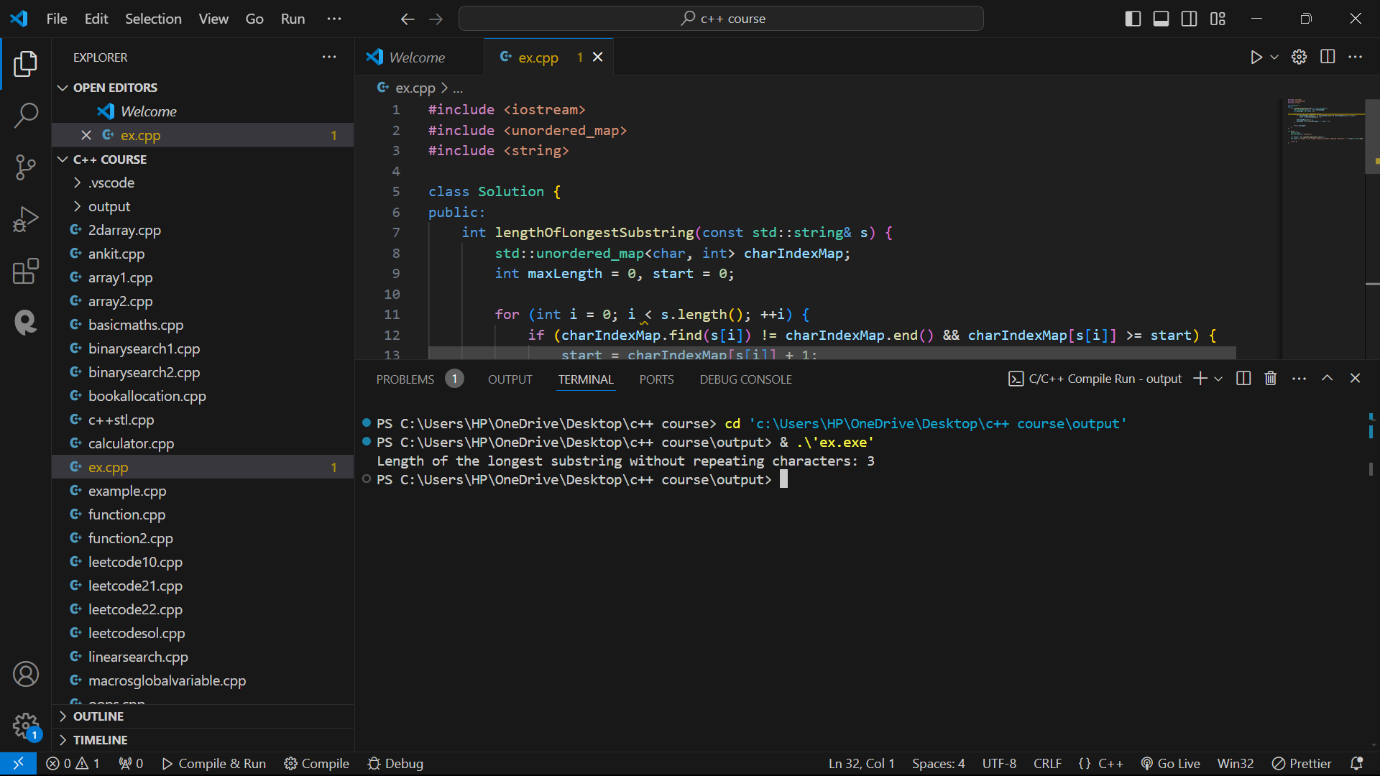
    int result = sol.lengthOfLongestSubstring(str);

    std::cout << "Length of the longest substring without repeating characters: " << result << 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.

Solution:

#include <iostream>

#include <climits>

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;

        maxPathSumHelper(root, maxSum);

        return maxSum;

    }

private:

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

        if (node == nullptr) return 0;

        int leftMax = std::max(0, maxPathSumHelper(node->left, maxSum));

        int rightMax = std::max(0, maxPathSumHelper(node->right, maxSum));

        int currentMax = node->val + leftMax + rightMax;

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

        return node->val + std::max(leftMax, rightMax);

    }

};

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);

    Solution sol;

    int result = sol.maxPathSum(root);

    std::cout << "Maximum path sum: " << result << std::endl;

    delete root->right->right;

    delete root->right->left;

    delete root->right;

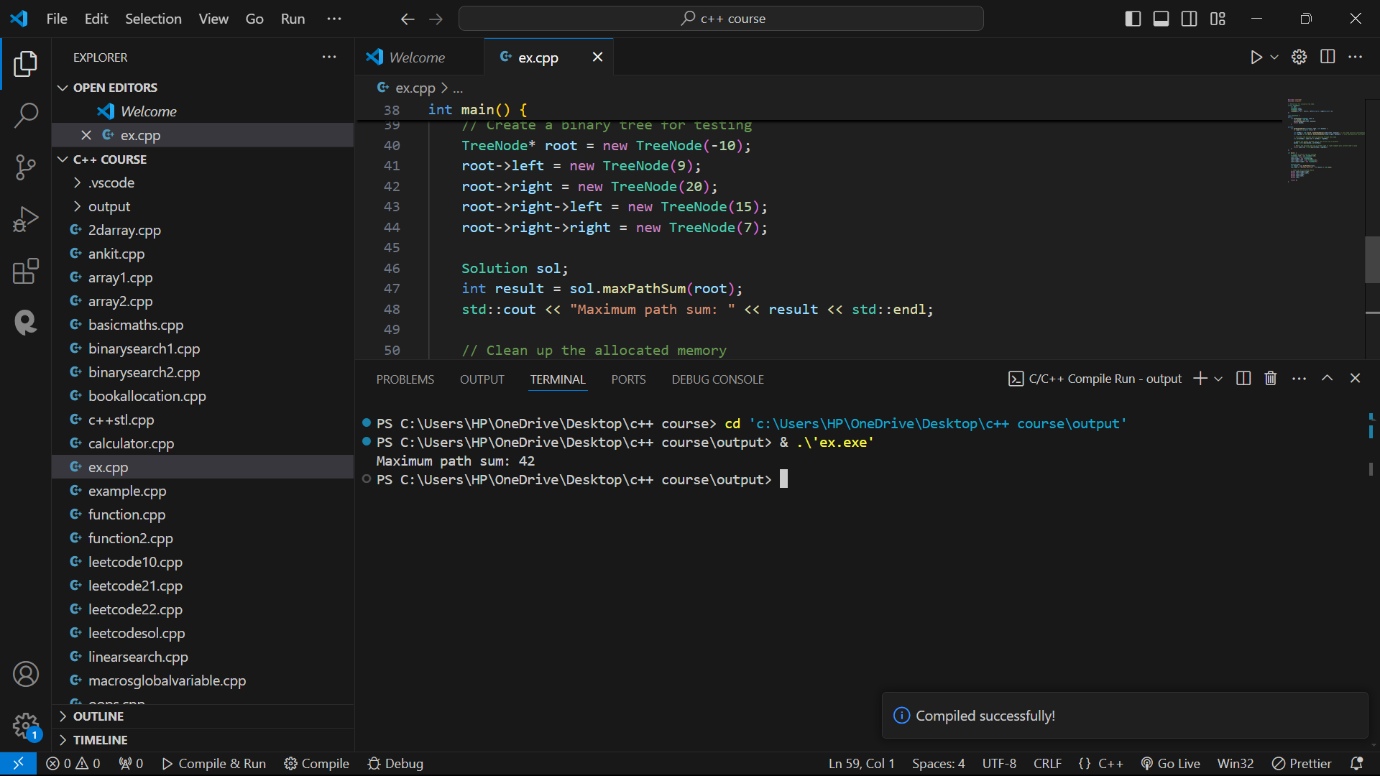
    delete root->left;

    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.

Solution:

#include <iostream>

#include <string>

#include <sstream>

#include <queue>

struct TreeNode {

    int val;

    TreeNode\* left;

    TreeNode\* right;

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

};

class Codec {

public:

    std::string serialize(TreeNode\* root) {

        if (!root) return "null";

        std::queue<TreeNode\*> q;

        q.push(root);

        std::stringstream ss;

        while (!q.empty()) {

            TreeNode\* node = q.front();

            q.pop();

            if (node) {

                ss << node->val << " ";

                q.push(node->left);

                q.push(node->right);

            } else {

                ss << "null ";

            }

        }

        return ss.str();

    }

    TreeNode\* deserialize(const std::string& data) {

        if (data == "null") return nullptr;

        std::stringstream ss(data);

        std::string item;

        std::getline(ss, item, ' ');

        TreeNode\* root = new TreeNode(std::stoi(item));

        std::queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            TreeNode\* node = q.front();

            q.pop();

            if (std::getline(ss, item, ' ')) {

                if (item != "null") {

                    node->left = new TreeNode(std::stoi(item));

                    q.push(node->left);

                }

            }

            if (std::getline(ss, item, ' ')) {

                if (item != "null") {

                    node->right = new TreeNode(std::stoi(item));

                    q.push(node->right);

                }

            }

        }

        return root;

    }

};

void printTree(TreeNode\* root) {

    if (!root) {

        std::cout << "null" << std::endl;

        return;

    }

    std::queue<TreeNode\*> q;

    q.push(root);

    while (!q.empty()) {

        TreeNode\* node = q.front();

        q.pop();

        if (node) {

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

            q.push(node->left);

            q.push(node->right);

        } else {

            std::cout << "null ";

        }

    }

    std::cout << std::endl;

}

int main() {

    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);

    Codec codec;

    std::string serialized = codec.serialize(root);

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

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

    std::cout << "Deserialized: ";

    printTree(deserialized);

    delete root->right->right;

    delete root->right->left;

    delete root->right;

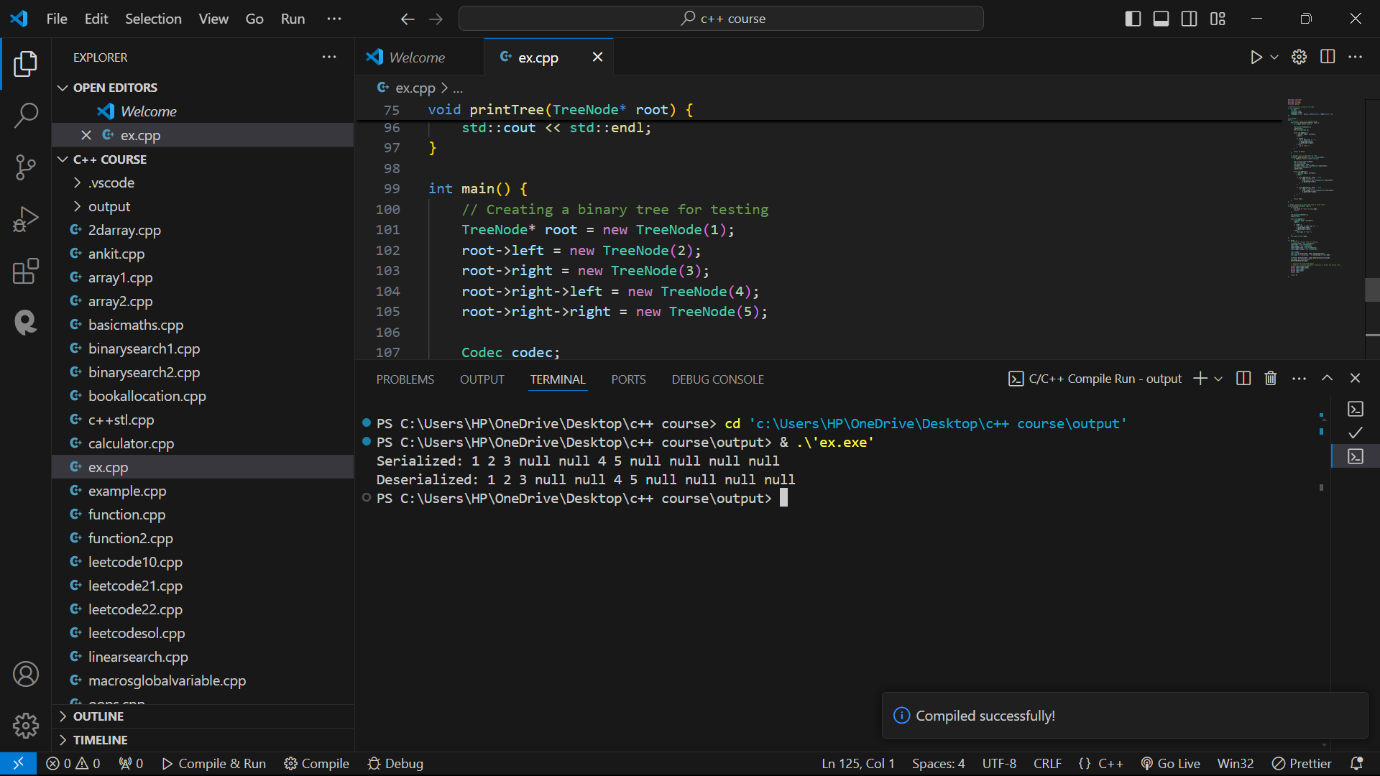
    delete root->left;

    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.

Solution:

#include <iostream>

#include <vector>

#include <algorithm>

class Solution {

public:

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

        int n = nums.size();

        k = k % n;

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

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

        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() {

    Solution sol;

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

    int k = 3;

    std::cout << "Original array: ";

    printArray(nums);

    sol.rotate(nums, k);

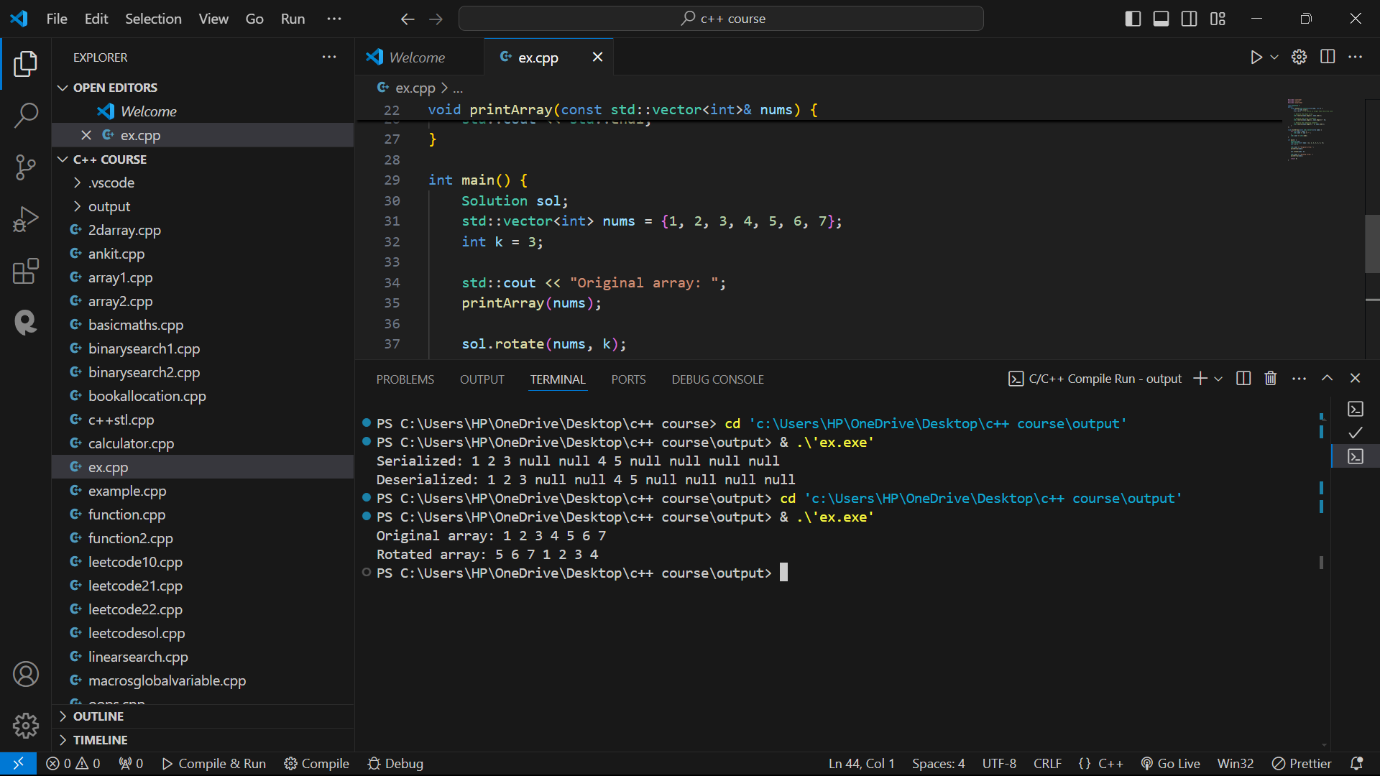
    std::cout << "Rotated array: ";

    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.

Solution:

#include <iostream>

class Solution {

public:

    int factorial(int n) {

        if (n < 0) {

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

            return -1;

        }

        int result = 1;

        for (int i = 1; i <= n; ++i) {

            result \*= i;

        }

        return result;

    }

};

int main() {

    Solution sol;

    int number;

    std::cout << "Enter a number: ";

    std::cin >> number;

    int result = sol.factorial(number);

    if (result != -1) {

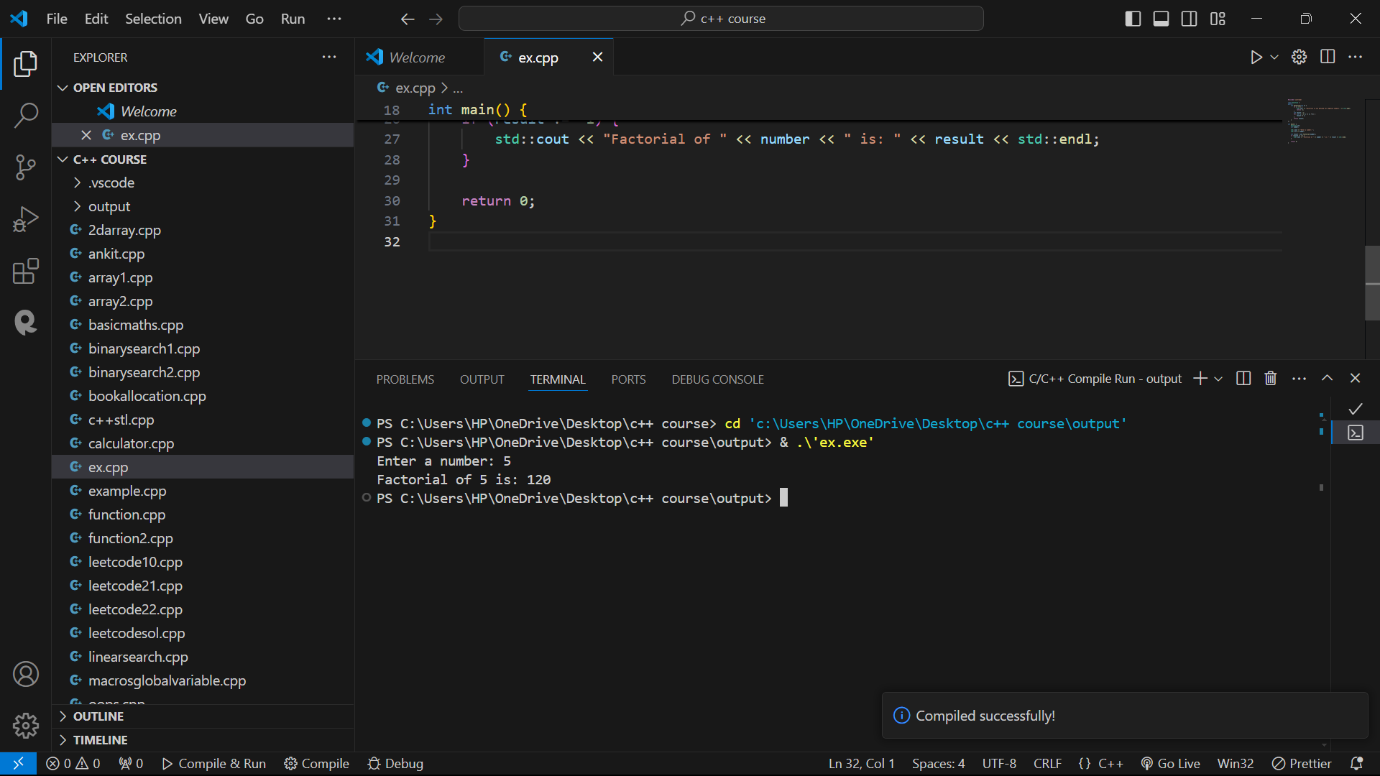
        std::cout << "Factorial of " << number << " is: " << result << std::endl;

    }

    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.

Solution:

#include <iostream>

class Solution {

public:

    int sumOfDigits(int num) {

        int sum = 0;

        while (num != 0) {

            sum += num % 10;

            num /= 10;

        }

        return sum;

    }

};

int main() {

    Solution sol;

    int number;

    std::cout << "Enter a number: ";

    std::cin >> number;

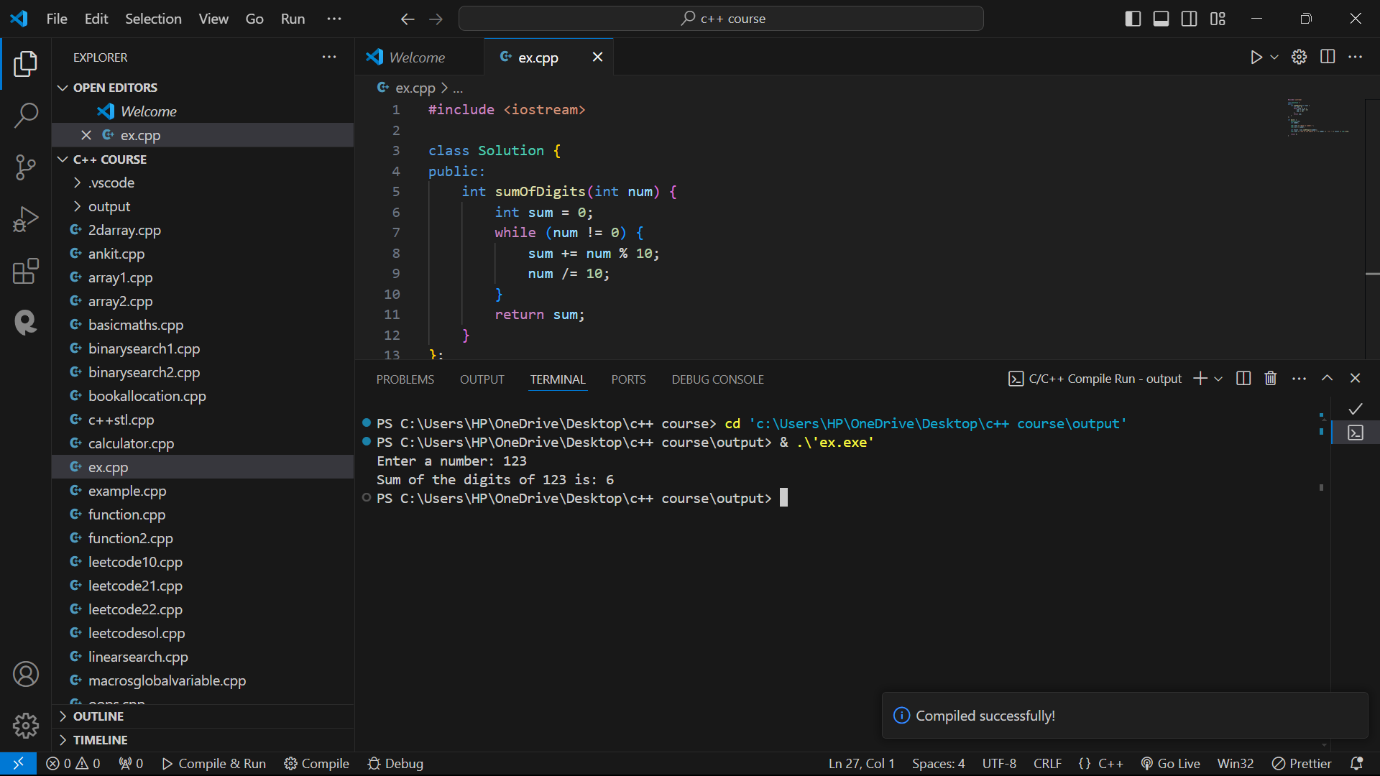
    int result = sol.sumOfDigits(number);

    std::cout << "Sum of the digits of " << number << " is: " << result << 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.

Solution:

#include <iostream>

class Solution {

public:

    int gcd(int a, int b) {

        while (b != 0) {

            int temp = b;

            b = a % b;

            a = temp;

        }

        return a;

    }

};

int main() {

    Solution sol;

    int a, b;

    std::cout << "Enter two numbers: ";

    std::cin >> a >> b;

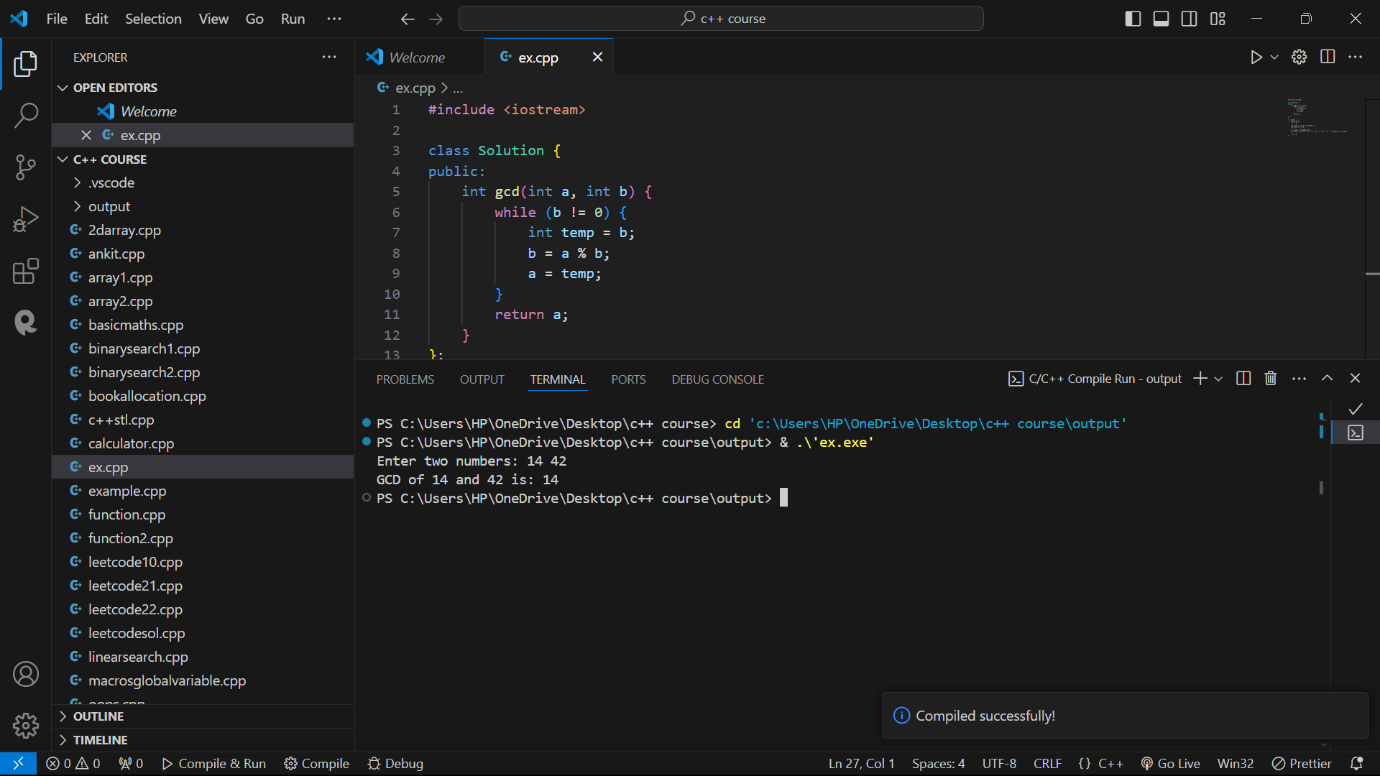
    int result = sol.gcd(a, b);

    std::cout << "GCD of " << a << " and " << b << " is: " << result << 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.

Solution:

#include <iostream>

#include <vector>

#include <algorithm>

#include <limits.h>

class Solution {

public:

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

        if (nums.size() < 2) {

            std::cerr << "Array should have at least two elements." << std::endl;

            return -1;

        }

        int minElement = INT\_MAX;

        int maxElement = INT\_MIN;

        for (int num : nums) {

            if (num < minElement) {

                minElement = num;

            }

            if (num > maxElement) {

                maxElement = num;

            }

        }

        return maxElement - minElement;

    }

};

int main() {

    Solution sol;

    std::vector<int> nums = {1, 2, 90, 10, 110};

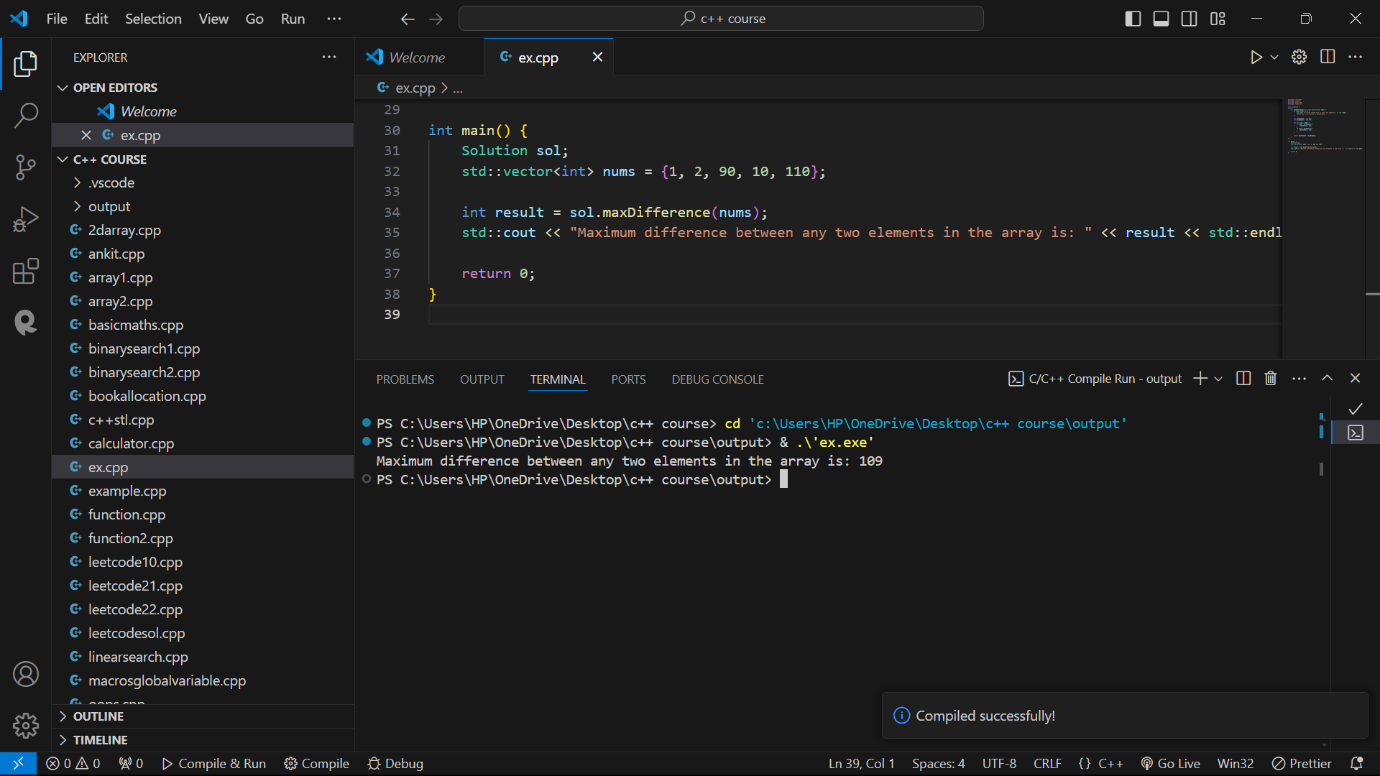
    int result = sol.maxDifference(nums);

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

    return 0;

}

Output:



Question – 10

Write a function to check if a given string contains only alphabetic characters.The function should return true if the string contains only alphabetic characters, and false otherwise.

Solution:

#include <iostream>

#include <string>

class Solution {

public:

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

        for (char c : str) {

            if (!isalpha(c)) {

                return false;

            }

        }

        return true;

    }

};

int main() {

    Solution sol;

    std::string input;

    std::cout << "Enter a string: ";

    std::cin >> input;

    bool result = sol.isAlphabetic(input);

    if (result) {

        std::cout << "The string contains only alphabetic characters." << std::endl;

    } else {

        std::cout << "The string contains non-alphabetic characters." << std::endl;

    }

    return 0;

}

Output:

