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.

#include <iostream>

using namespace std;

struct ListNode {

    int val;

    ListNode\* next;

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

};

ListNode\* reverseList(ListNode\* head) {

    ListNode\* prev = nullptr;

    ListNode\* curr = head;

    while (curr != nullptr) {

        ListNode\* nextTemp = curr->next;

        curr->next = prev;

        prev = curr;

        curr = nextTemp;

    }

    return prev;

}

void printList(ListNode\* head) {

    ListNode\* curr = head;

    while (curr != nullptr) {

        cout << curr->val << " ";

        curr = curr->next;

    }

    cout << 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);

    cout << "Original list: ";

    printList(head);

    ListNode\* newHead = reverseList(head);

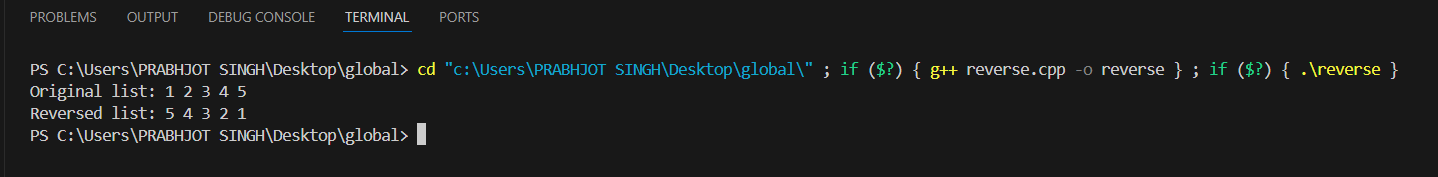
    cout << "Reversed list: ";

    printList(newHead);

    return 0;

}

Output:-



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.

#include <iostream>

#include <unordered\_map>

#include <string>

using namespace std;

int lengthOfLongestSubstring(string s) {

    unordered\_map<char, int> charIndexMap;

    int maxLength = 0;

    int start = 0;

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

        char currentChar = s[end];

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

            start = charIndexMap[currentChar] + 1;

        }

        charIndexMap[currentChar] = end;

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

    }

    return maxLength;

}

int main() {

    string s;

    cout << "Enter a string: ";

    cin >> s;

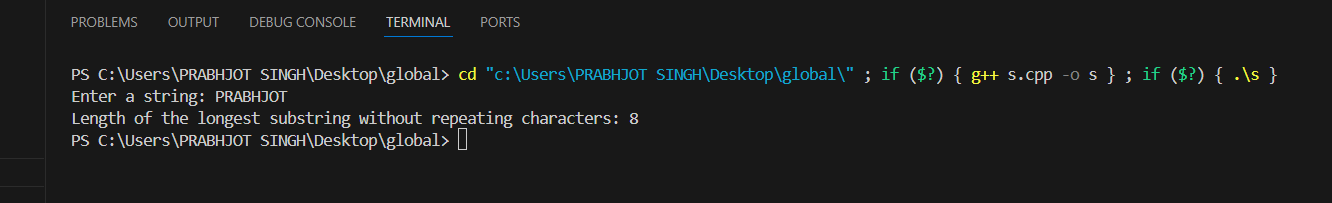
    int result = lengthOfLongestSubstring(s);

    cout << "Length of the longest substring without repeating characters: " << result << endl;

    return 0;

}

Output:-



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.

#include <iostream>

#include <algorithm>

#include <climits>

using namespace std;

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 globalMax = INT\_MIN;

        maxPathSumHelper(root, globalMax);

        return globalMax;

    }

private:

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

        if (!node) return 0;

        int leftMax = max(maxPathSumHelper(node->left, globalMax), 0);

        int rightMax = max(maxPathSumHelper(node->right, globalMax), 0);

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

        globalMax = max(globalMax, currentMax);

        return node->val + 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 solution;

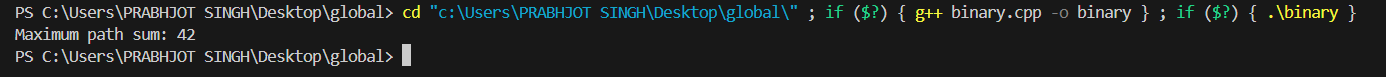
    int result = solution.maxPathSum(root);

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

    return 0;

}

Output:-



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.

#include <iostream>

#include <sstream>

#include <string>

#include <queue>

using namespace std;

struct TreeNode {

    int val;

    TreeNode\* left;

    TreeNode\* right;

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

};

class Codec {

public:

    string serialize(TreeNode\* root) {

        ostringstream out;

        serializeHelper(root, out);

        return out.str();

    }

    TreeNode\* deserialize(const string& data) {

        istringstream in(data);

        return deserializeHelper(in);

    }

private:

    void serializeHelper(TreeNode\* node, ostringstream& out) {

        if (node) {

            out << node->val << ' ';

            serializeHelper(node->left, out);

            serializeHelper(node->right, out);

        } else {

            out << "# ";

        }

    }

    TreeNode\* deserializeHelper(istringstream& in) {

        string val;

        in >> val;

        if (val == "#") {

            return nullptr;

        } else {

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

            node->left = deserializeHelper(in);

            node->right = deserializeHelper(in);

            return node;

        }

    }

};

void printTree(TreeNode\* root) {

    if (!root) return;

    queue<TreeNode\*> q;

    q.push(root);

    while (!q.empty()) {

        TreeNode\* current = q.front();

        q.pop();

        if (current) {

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

            q.push(current->left);

            q.push(current->right);

        } else {

            cout << "# ";

        }

    }

    cout << 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;

    string serializedData = codec.serialize(root);

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

    TreeNode\* deserializedTree = codec.deserialize(serializedData);

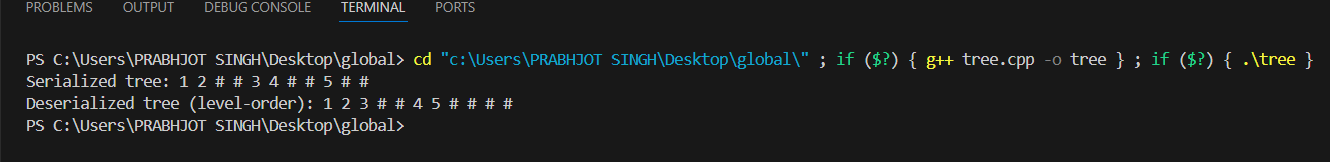
    cout << "Deserialized tree (level-order): ";

    printTree(deserializedTree);

    return 0;

}

Output :-



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.

#include <iostream>

#include <algorithm>

using namespace std;

void reverse(int arr[], int start, int end) {

    while (start < end) {

        swap(arr[start], arr[end]);

        start++;

        end--;

    }

}

void rotate(int arr[], int n, int k) {

    k = k % n;

    if (k == 0) return;

    reverse(arr, 0, n - 1);

    reverse(arr, 0, k - 1);

    reverse(arr, k, n - 1);

}

int main() {

    int arr[] = {1, 2, 3, 4, 5, 6, 7,8};

    int n = sizeof(arr) / sizeof(arr[0]);

    int k = 3;

    cout << "Original array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

    rotate(arr, n, k);

    cout << "Rotated array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

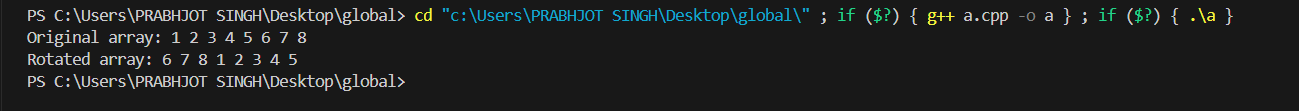
    }

    cout << endl;

    return 0;

}

Output :-



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

factorial of the number.

#include <iostream>

using namespace std;

int factorial(int n)

{

    if (n == 0 || n == 1)

        return 1;

    return n \* factorial(n - 1);

}

int main()

{

    int num = 5;

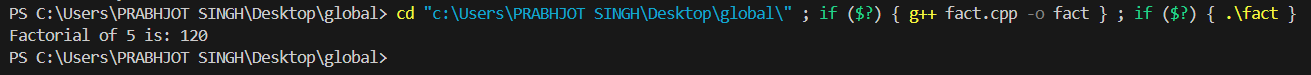
    cout << "Factorial of "

        << num << " is: " << factorial(num) << endl;

    return 0;

}

Output :-



7. Write a function to compute the sum of the digits of a given number.T he function should

return the sum of the digits of the number.

# include <iostream>

using namespace std;

int sum\_of\_digits(int n)

{

return n == 0 ? 0 : n%10 + sum\_of\_digits(n/10) ;

}

int main()

{

int n;

cout << "Enter a number : ";

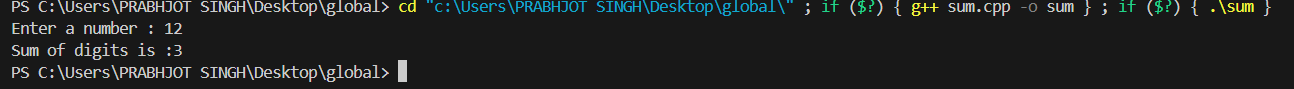
cin >> n;

cout << "Sum of digits is :12"<< sum\_of\_digits(n) << endl;

return 0;

}

Output :-



8. Write a function to find the greatest common divisor (GCD) of two numbers. The function

should return the GCD of a and b.

#include <iostream>

using namespace std;

int gcd(int a, int b) {

    while (b != 0) {

        int temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

int main() {

    int a, b;

    cout << "Enter two numbers: ";

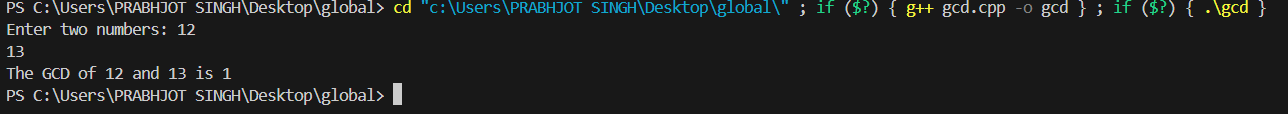
    cin >> a >> b;

    cout << "The GCD of " << a << " and " << b << " is " << gcd(a, b) << endl;

    return 0;

}

Output :-



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.

#include<iostream>

using namespace std;

int maxDifference(int arr[], int size) {

    if (size < 2)

        return 0;

    int minVal = arr[0];

    int maxDiff = 0;

    for (int i = 1; i < size; ++i) {

        if (arr[i] < minVal)

            minVal = arr[i];

        else if (arr[i] - minVal > maxDiff)

            maxDiff = arr[i] - minVal;

    }

    return maxDiff;

}

int main() {

    int nums[] = {2, 7, 9, 5, 1, 3, 5};

    int size = sizeof(nums) / sizeof(nums[0]);

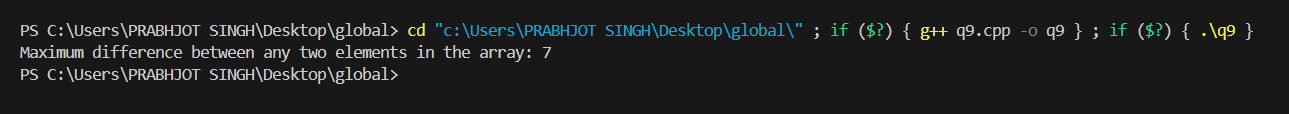
    int maxDiff = maxDifference(nums, size);

    cout<< "Maximum difference between any two elements in the array: " << maxDiff <<endl;

    return 0;

}

Output:-



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.

#include <iostream>

#include <string>

using namespace std;

bool alphabetic(const string& str) {

    for (char c : str) {

        if (!isalpha(c)) {

            return false;

        }

    }

    return true;

}

int main() {

    string input;

    cout << "Enter a string: ";

    getline(cin, input);

    if (alphabetic(input)) {

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

    } else {

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

    }

    return 0;

}

Output :-

