**Global Trend Programming Profile Assessment Questions**

Welcome to the Global Trend Programming Profile assessment. This document outlines the requirements and guidelines for completing the programming assessment. Please read the instructions carefully before you begin.

## Assessment Overview

This assessment consists of 10 programming questions designed to evaluate your problem-solving and coding skills in C++. You are required to attempt all the questions within a 1-day time period. The assessment is timed, and your submissions will be evaluated based on correctness, efficiency, and coding best practices.

Github Link: https://github.com/Ashutosh8668/GlobalTrend\_Programming\_Assessment

**Questions**

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 <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;

ListNode\* nextNode = nullptr;

while (curr != nullptr) {

nextNode = curr->next;

curr->next = prev;

prev = curr;

curr = nextNode;

}

return prev;

}

void printList(ListNode\* head) {

ListNode\* temp = head;

while (temp != nullptr) {

cout << temp->val << " ";

temp = temp->next;

}

cout << endl;

}

void push(ListNode\*\* head\_ref, int new\_data) {

ListNode\* new\_node = new ListNode(new\_data);

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

int main() {

ListNode\* head = nullptr;

push(&head, 5);

push(&head, 4);

push(&head, 3);

push(&head, 2);

push(&head, 1);

cout << "Original Linked List: ";

printList(head);

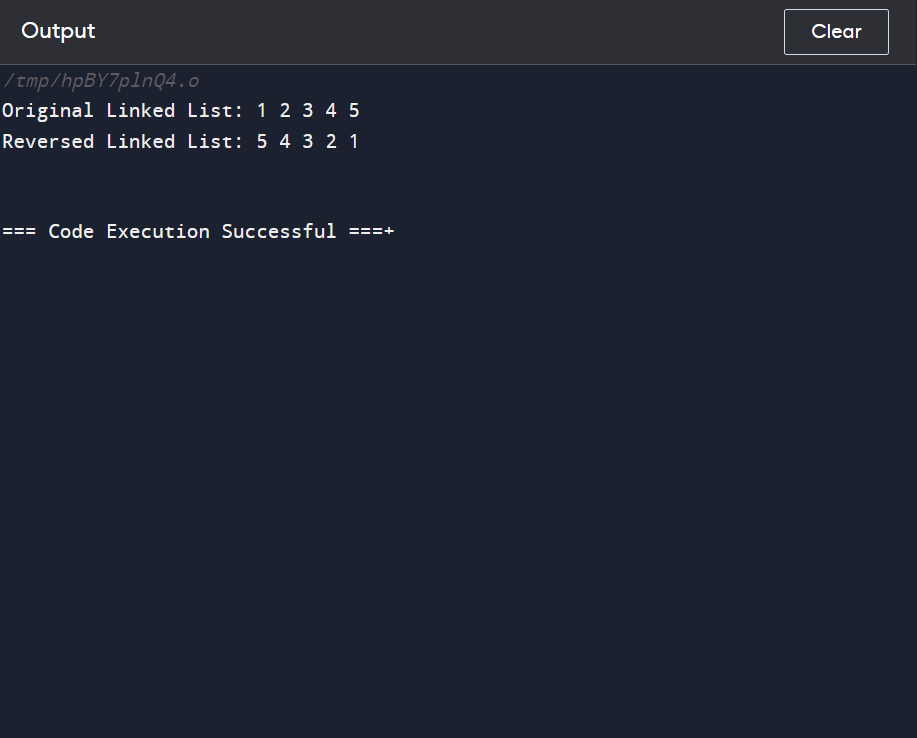
head = reverseList(head);

cout << "Reversed Linked List: ";

printList(head);

return 0;

}



1. **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>

using namespace std;

int lengthOfLongestSubstring(string s) {

unordered\_map<char, int> charIndex;

int maxLen = 0, start = 0;

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

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

start = max(start, charIndex[s[end]] + 1);

}

charIndex[s[end]] = end;

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

}

return maxLen;

}

int main() {

string s = "abcabcbb";

cout << "Length of longest substring without repeating characters in \"" << s << "\": " << lengthOfLongestSubstring(s) << endl;

s = "bbbbb";

cout << "Length of longest substring without repeating characters in \"" << s << "\": " << lengthOfLongestSubstring(s) << endl;

s = "pwwkew";

cout << "Length of longest substring without repeating characters in \"" << s << "\": " << lengthOfLongestSubstring(s) << endl;

s = "";

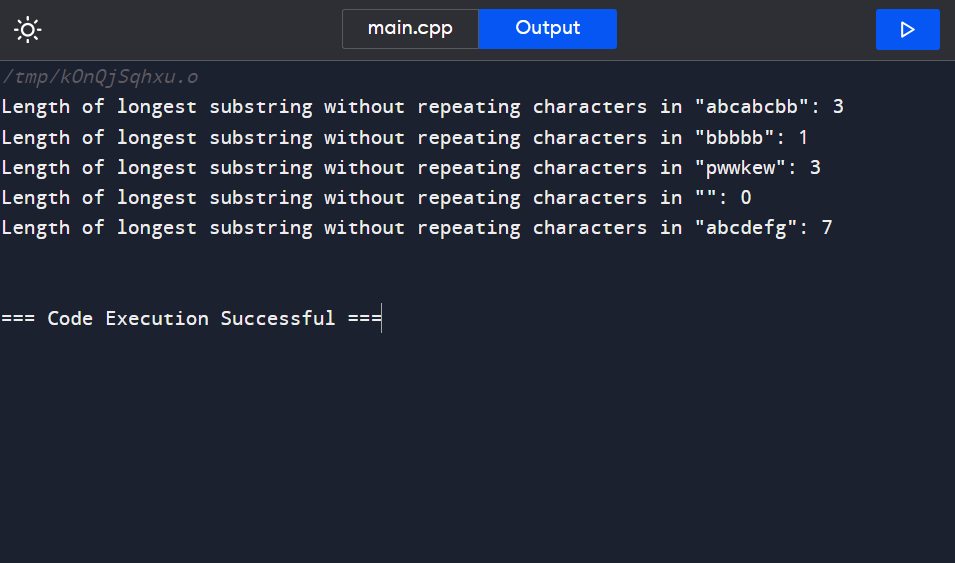
cout << "Length of longest substring without repeating characters in \"" << s << "\": " << lengthOfLongestSubstring(s) << endl;

s = "abcdefg";

cout << "Length of longest substring without repeating characters in \"" << s << "\": " << lengthOfLongestSubstring(s) << endl;

return 0;

}

****

1. **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 <sstream>

#include <queue>

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 "null";

stringstream ss;

queue<TreeNode\*> q;

q.push(root);

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();

}

// Decodes your encoded data to tree

TreeNode\* deserialize(string data) {

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

stringstream ss(data);

string str;

ss >> str;

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

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

if (!(ss >> str)) break;

if (str != "null") {

node->left = new TreeNode(stoi(str));

q.push(node->left);

}

if (!(ss >> str)) break;

if (str != "null") {

node->right = new TreeNode(stoi(str));

q.push(node->right);

}

}

return root;

}

};

// Helper function to print the tree (Level Order)

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 << "null ";

}

}

cout << endl;

}

int main() {

// Creating a binary 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);

Codec ser, deser;

string serializedTree = ser.serialize(root);

cout << "Serialized Tree: " << serializedTree << endl;

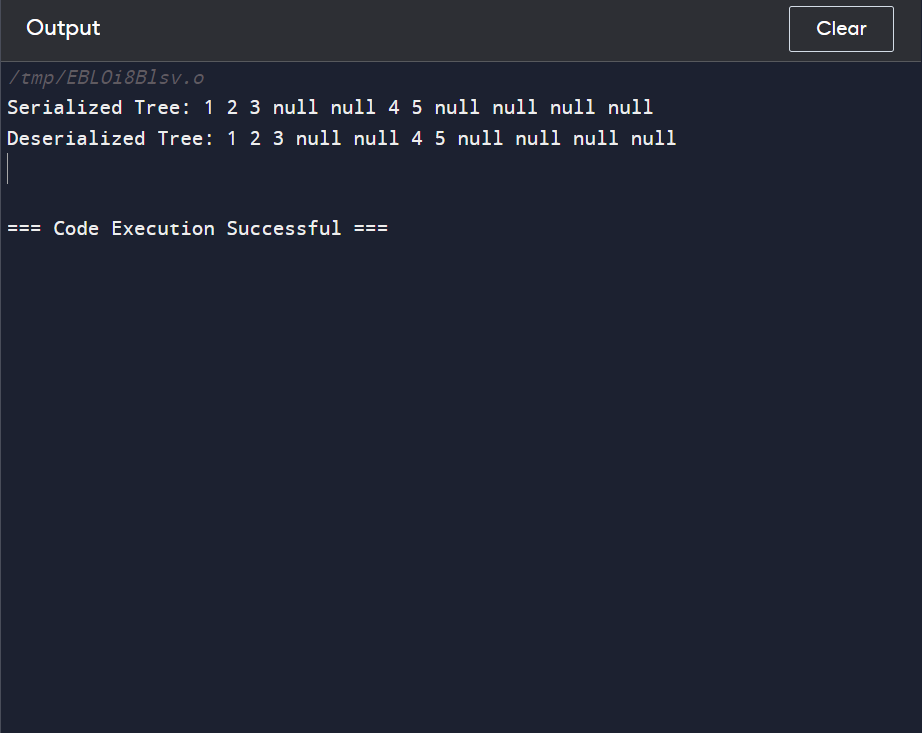
TreeNode\* deserializedTree = deser.deserialize(serializedTree);

cout << "Deserialized Tree: ";

printTree(deserializedTree);

return 0;

}



1. **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>

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 "null";

stringstream ss;

queue<TreeNode\*> q;

q.push(root);

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();

}

// Decodes your encoded data to tree

TreeNode\* deserialize(string data) {

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

stringstream ss(data);

string str;

ss >> str;

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

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

if (!(ss >> str)) break;

if (str != "null") {

node->left = new TreeNode(stoi(str));

q.push(node->left);

}

if (!(ss >> str)) break;

if (str != "null") {

node->right = new TreeNode(stoi(str));

q.push(node->right);

}

}

return root;

}

};

// Helper function to print the tree (Level Order)

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 << "null ";

}

}

cout << endl;

}

int main() {

// Creating a binary 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);

Codec ser, deser;

string serializedTree = ser.serialize(root);

cout << "Serialized Tree: " << serializedTree << endl;

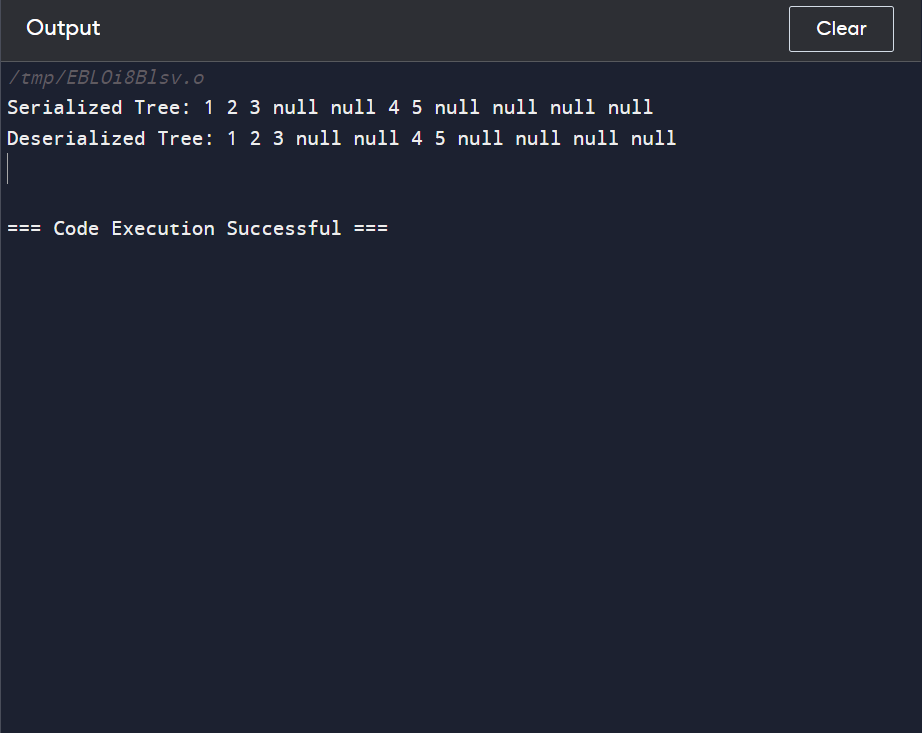
TreeNode\* deserializedTree = deser.deserialize(serializedTree);

cout << "Deserialized Tree: ";

printTree(deserializedTree);

return 0;

}

****

1. **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>

using namespace std;

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

int size = nums.size();

k = k % size;

if (k == 0) return;

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

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

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

}

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

for (int num : nums) {

cout << num << " ";

}

cout << endl;

}

int main() {

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

int steps = 3;

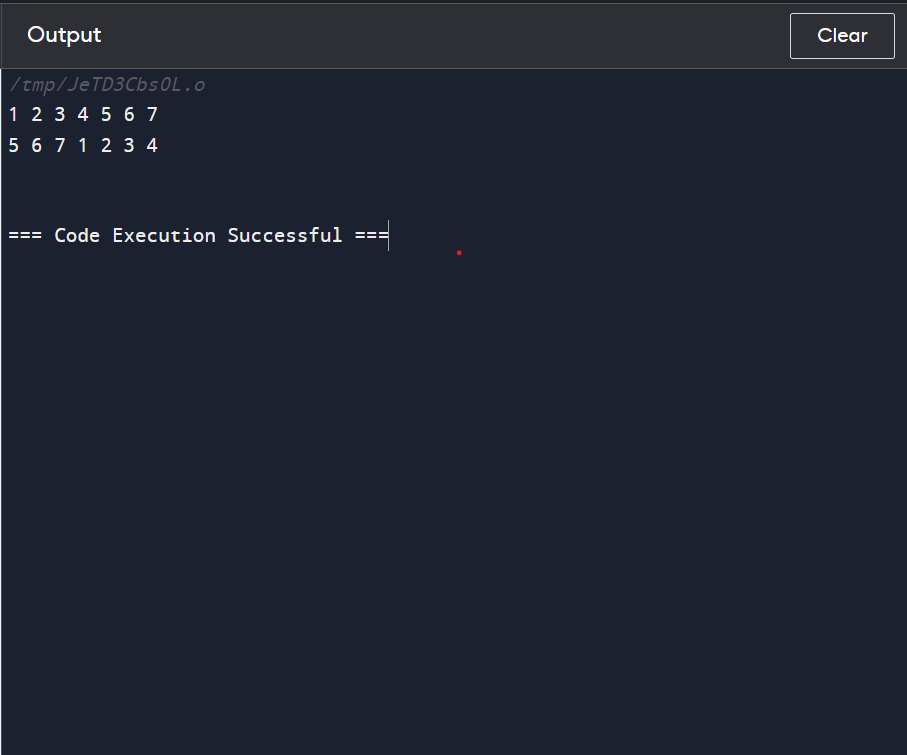
printArray(arr);

rotate(arr, steps);

printArray(arr);

return 0;

}



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

**#code:**

#include <iostream>

using namespace std;

unsigned long long factorial(int n) {

if (n < 0) {

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

return 0;

}

unsigned long long res = 1;

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

res \*= i;

}

return res;

}

int main() {

int num;

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

cin >> num;

unsigned long long result = factorial(num);

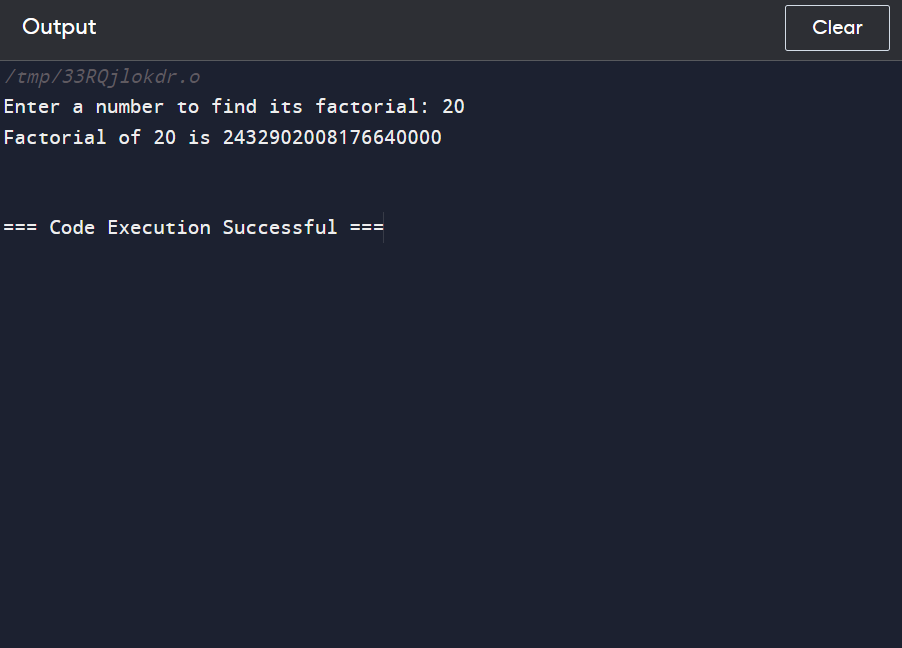
if (num >= 0) {

cout << "Factorial of " << num << " is " << result << endl;

}

return 0;

}



1. **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>

using namespace std;

int sumOfDigits(int num) {

int sum = 0;

if (num < 0) {

num = -num;

}

while (num > 0) {

sum += num % 10;

num /= 10;

}

return sum;

}

int main() {

int n;

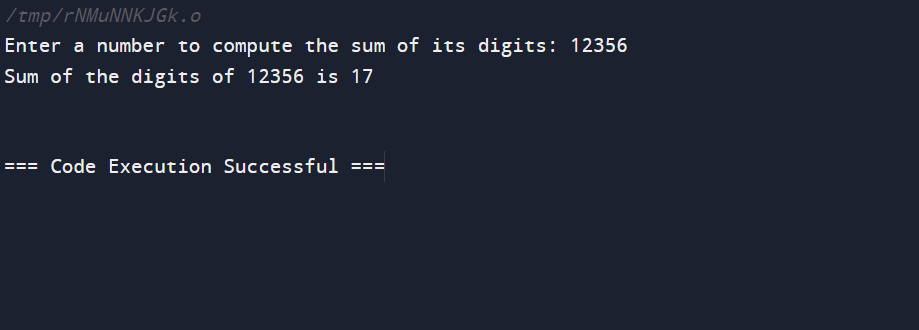
cout << "Enter a number to compute the sum of its digits: ";

cin >> n;

int res = sumOfDigits(n);

cout << "Sum of the digits of " << n << " is " << res << endl;

return 0;

}

1. **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 <iostream>

using namespace std;

int gcd(int x, int y) {

if (x < 0) x = -x;

if (y < 0) y = -y;

while (y != 0) {

int temp = y;

y = x % y;

x = temp;

}

return x;

}

int main() {

int num1, num2;

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

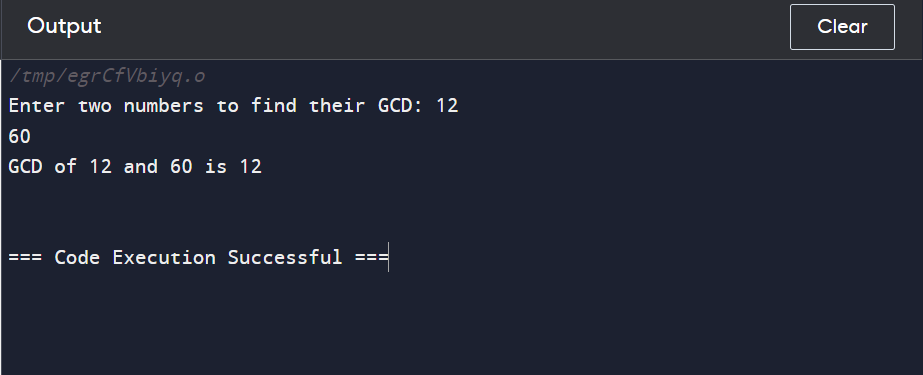
cin >> num1 >> num2;

int res = gcd(num1, num2);

cout << "GCD of " << num1 << " and " << num2 << " is " << res << endl;

return 0;

}



1. **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 <iostream>

#include <vector>

#include <algorithm>

using namespace std;

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

if (arr.size() < 2) {

cout << "Array must contain at least two elements." << endl;

return 0;

}

int minVal = \*min\_element(arr.begin(), arr.end());

int maxVal = \*max\_element(arr.begin(), arr.end());

return maxVal - minVal;

}

int main() {

vector<int> numbers = {2, 3, 10, 6, 4, 8, 1};

cout << "Array: ";

for (int num : numbers) {

cout << num << " ";

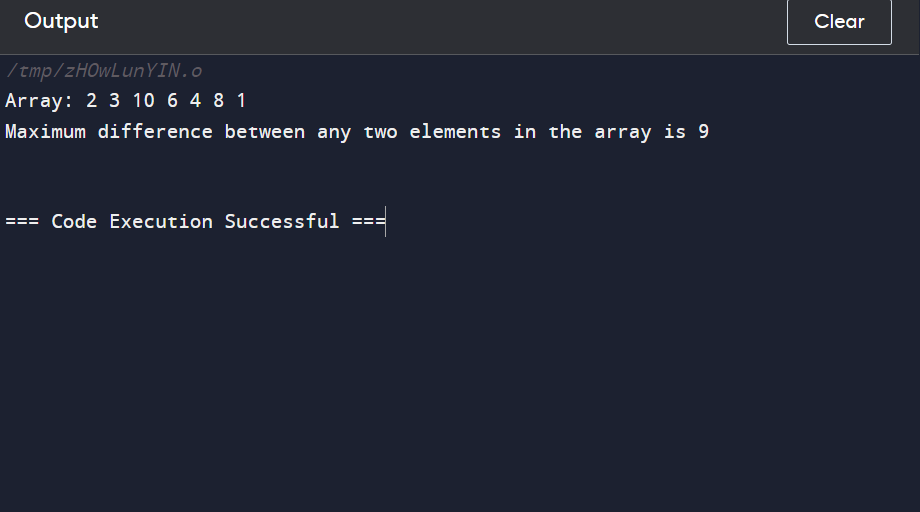
}

cout << endl;

int res = maxDifference(numbers);

cout << "Maximum difference between any two elements in the array is " << res << endl;

return 0;

}

1. **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.**

**#code:**

#include <iostream>

#include <string>

#include <cctype> // For using isalpha function

using namespace std;

bool containsOnlyAlpha(const string& s) {

for (char ch : s) {

if (!isalpha(ch)) {

return false;

}

}

return true;

}

int main() {

string str;

cout << "Enter a string to check if it contains only alphabetic characters: ";

getline(cin, str);

bool res = containsOnlyAlpha(str);

if (res) {

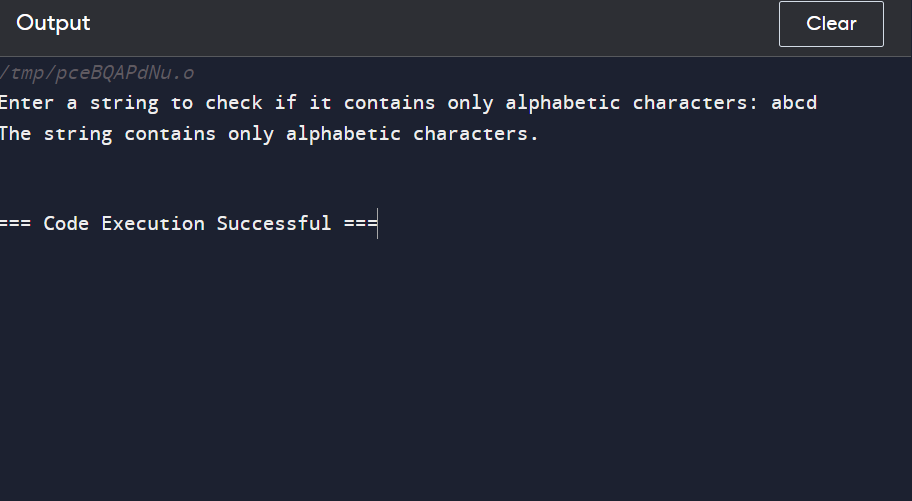
cout << "The string contains only alphabetic characters." << endl;

} else {

cout << "The string does not contain only alphabetic characters." << endl;

} return 0;

}



### 