**Global Trend Programming Profile Assessment Questions**

**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.

#include <iostream>

*// Definition for singly-linked list node*

struct ListNode {

int val;

ListNode \*next;

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

};

*// Function to reverse a singly linked list*

ListNode\* reverseList(ListNode\* *head*) {

ListNode \*prev = nullptr, \*next = nullptr; *// Initialize pointers prev and next to nullptr*

while (*head*) { *// Loop until head is nullptr*

next = *head*->next; *// Store the next node in the original list*

*head*->next = prev; *// Reverse the current node's next pointer to point to prev*

rev = *head*; *// Move prev pointer to current node*

*head* = next; *// Move head pointer to next node in the original list*

}

return prev; *// Return the new head of the reversed list*

}

*// Utility function to print the linked list*

void printList(ListNode\* *head*) {

ListNode\* temp = *head*; *// Initialize a temporary pointer to iterate through the list*

while (temp != nullptr) { *// Loop until temp is nullptr*

std::cout << temp->val << " "; *// Print the value of the current node*

temp = temp->next; *// Move temp pointer to the next node*

}

std::cout << std::endl; *// Print a newline after printing all nodes*

}

*// Function to create a linked list from user input*

ListNode\* createLinkedList() {

ListNode\* dummy = new ListNode(0); *// Create a dummy node to simplify list construction*

ListNode\* current = dummy; *// Initialize current pointer to the dummy node*

int n; *// Variable to store number of nodes*

std::cout << "Enter number of nodes: "; *// Prompt user to enter number of nodes*

std::cin >> n; *// Read number of nodes from input*

std::cout << "Enter values for each node:" << std::endl; *// Prompt user to enter values for each node*

for (int i = 0; i < n; ++i) { *// Loop to read values for each node*

int value; *// Variable to store node value*

std::cin >> value; *// Read node value from input*

current->next = new ListNode(value); *// Create a new node with the read value and link it to current node*

current = current->next; *// Move current pointer to the newly created node*

}

return dummy->next; *// Return the head of the created linked list (excluding the dummy node)*

}

*// Test example*

int main() {

ListNode\* head = createLinkedList(); *// Create a linked list from user input*

std::cout << "Original list: ";

printList(head); *// Print the original list*

ListNode\* reversedHead = reverseList(head); *// Reverse the list and get the new head of reversed list*

std::cout << "Reversed list: ";

printList(reversedHead); *// Print the reversed list*

return 0; *// Exit the program*

}

**Output**

Enter number of nodes: 5

Enter values for each node:

868

9

5

4

Original list: 8 86 9 5 4

Reversed list: 4 5 9 86 8

=== Code Execution Successful ===

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.

#include <iostream>

#include <unordered\_map>

#include <algorithm>

*// Function to find the length of the longest substring without repeating characters*

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

    int n = *s*.length(); *// Get the length of the input string*

    std::unordered\_map<char, int> charMap; *// Hash map to store characters and their indices*

    int maxLength = 0; *// Variable to store the maximum length of substring without repeating characters*

    int left = 0; *// Left pointer of the sliding window*

    for (int right = 0; right < n; ++right) { *// Iterate through the string with the right pointer*

        if (charMap.find(*s*[right]) != charMap.end()) { *// If current character is already in the map*

            left = std::max(charMap[*s*[right]] + 1, left); *// Move left pointer to avoid repeating characters*

        }

        charMap[*s*[right]] = right; *// Update the index of current character in the map*

        maxLength = std::max(maxLength, right - left + 1); *// Update maxLength with the current window size*

    }

    return maxLength; *// Return the maximum length of substring without repeating characters*

}

*// Test example*

int main() {

    std::string input;

    std::cout << "Enter a string: ";

    std::cin >> input; *// User inputs a string*

    int result = lengthOfLongestSubstring(input); *// Calculate the length of longest substring without repeating characters*

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

              << result << std::endl; *// Output the result*

    return 0; *// Exit the program*

}

**Output**

Enter a string: rgtdffsdcx

Length of the longest substring without repeating characters: 5

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.

#include <iostream>

#include <algorithm> *// For std::max*

*// Definition for a binary tree node*

struct TreeNode {

    int val;

    TreeNode\* left;

    TreeNode\* right;

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

};

*// Helper function to recursively calculate the maximum path sum*

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

      if (!*root*) return 0; *// Base case: if root is nullptr, return 0*

      int leftSum = std::max(0, maxPathSumHelper(*root*->left, *maxSum*)); *// Recursively calculate max path sum from left subtree*

      int rightSum = std::max(0, maxPathSumHelper(*root*->right, *maxSum*)); *// Recursively calculate max path sum from right subtree*

      int currentSum = *root*->val + leftSum + rightSum; *// Calculate the current path sum through the root*

*maxSum* = std::max(*maxSum*, currentSum); *// Update maxSum with the maximum of currentSum and maxSum*

      return *root*->val + std::max(leftSum, rightSum); *// Return the maximum path sum possible from root to either left or right subtree*

}

*// Function to find the maximum path sum in a binary tree*

int maxPathSum(TreeNode\* *root*) {

      int maxSum = INT\_MIN; *// Initialize maxSum to the minimum integer value*

      maxPathSumHelper(*root*, maxSum); *// Call helper function to recursively calculate max path sum*

      return maxSum; *// Return the final maximum path sum*

}

*// Function to create a binary tree from user input (for testing purposes)*

TreeNode\* createTree() {

      int value;

      std::cout << "Enter node value (-1 for null): ";

      std::cin >> value;

      if (value == -1) {

         return nullptr;

      }

      TreeNode\* root = new TreeNode(value);

      std::cout << "Enter left child of " << value << ":" << std::endl;

      root->left = createTree();

      std::cout << "Enter right child of " << value << ":" << std::endl;

      root->right = createTree();

      return root;

}

*// Test example*

int main() {

      TreeNode\* root = createTree(); *// Create a binary tree from user input*

      int result = maxPathSum(root); *// Find the maximum path sum in the tree*

      std::cout << "Maximum path sum in the binary tree: " << result << std::endl; *// Output the result*

      return 0; *// Exit the program*

}

**Output**

Enter node value (-1 for null): 1

Enter left child of 1:

Enter node value (-1 for null): 2

Enter left child of 2:

Enter node value (-1 for null): -1

Enter right child of 2:

Enter node value (-1 for null): -1

Enter right child of 1:

Enter node value (-1 for null): 3

Enter left child of 3:

Enter node value (-1 for null): -1

Enter right child of 3:

Enter node value (-1 for null): 4

Enter left child of 4:

Enter node value (-1 for null): -1

Enter right child of 4:

Enter node value (-1 for null): -1

Maximum path sum in the binary tree: 10

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.

#include <iostream>

#include <sstream> *// For std::ostringstream and std::istringstream*

#include <string>  *// For std::string*

*// Definition for a binary tree node*

struct TreeNode {

    int val;

    TreeNode\* left;

    TreeNode\* right;

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

};

*// Serialize the binary tree to a string*

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

    if (!*root*) {

        return "null,"; *// If node is null, represent it as "null"*

    }

*// Convert the current node to string and serialize left and right subtrees recursively*

    return std::to\_string(*root*->val) + ","

         + serialize(*root*->left)

         + serialize(*root*->right);

}

*// Deserialize the string to reconstruct the binary tree*

TreeNode\* deserialize(std::istringstream& *ss*) {

    std::string token;

    std::getline(ss, token, ','); *// Read next token separated by ','*

    if (token == "null") {

        return nullptr; *// If token is "null", return nullptr indicating null node*

    }

*// Convert token to integer (node value) and create a new TreeNode*

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

    root->left = deserialize(ss); *// Deserialize left subtree recursively*

    root->right = deserialize(ss); *// Deserialize right subtree recursively*

    return root; *// Return the constructed TreeNode*

}

*// Test example*

int main() {

*// Example tree creation for testing purposes*

    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 binary tree*

    std::string serializedTree = serialize(root);

    std::cout << "Serialized tree: " << serializedTree << std::endl;

*// Deserialize the string to reconstruct the binary tree*

    std::istringstream iss(serializedTree);

    TreeNode\* reconstructedTree = deserialize(iss);

*// Clean up memory (optional, depending on application)*

*// In a real application, you would handle memory management more carefully*

    delete root->left;

    delete root->right->left;

    delete root->right->right;

    delete root->right;

    delete root;

    return 0; *// Exit the program*

}

**Output**

Serialized tree: 1,2,null,null,3,4,null,null,5,null,null,

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.

#include <iostream>

#include <vector>

#include <algorithm> *// Include for std::reverse*

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

    int n = nums.size();

    k = k % n; *// Adjust k in case it's larger than the array size*

*// Reverse the entire array*

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

*// Reverse the first k elements*

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

*// Reverse the rest of the elements*

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

}

*// Test example*

int main() {

    std::vector<int> nums;

    int n, k;

*// Input size of array*

    std::cout << "Enter number of elements: ";

    std::cin >> n;

*// Input array elements*

    std::cout << "Enter elements of array: ";

    for (int i = 0; i < n; ++i) {

        int num;

        std::cin >> num;

        nums.push\_back(num);

    }

*// Input number of steps to rotate*

    std::cout << "Enter number of steps to rotate: ";

    std::cin >> k;

    rotate(nums, k); *// Rotate the array in-place*

*// Output the rotated array*

    std::cout << "Rotated array: ";

    for (int num : nums) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

    return 0; *// Exit the program*

}

**Output**

Enter number of elements: 7

Enter elements of array: 2 9 4 6 3 4 5

Enter number of steps to rotate: 4

Rotated array: 6 3 4 5 2 9 4

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

#include <iostream>

*// Function to calculate the factorial of a number*

int factorial(int *n*) {

    int result = 1; *// Initialize result to 1, as factorial of 0 and 1 is 1*

*// Calculate factorial iteratively*

    for (int i = 2; i <= *n*; ++i) {

        result \*= i; *// Multiply result by current value of i*

    }

    return result; *// Return the calculated factorial*

}

*// Test example*

int main() {

    int n;

*// Input number for which factorial is to be calculated*

    std::cout << "Enter a number: ";

    std::cin >> n;

*// Calculate factorial of n*

    int result = factorial(n);

*// Output the factorial*

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

    return 0; *// Exit the program*

}

**Output**

Enter a number: 5

Factorial of 5 is: 120

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.

#include <iostream>

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

int sumOfDigits(int *n*) {

    int sum = 0; *// Initialize sum to 0*

*// Iterate through each digit of the number*

    while (*n* > 0) {

        sum += *n* % 10; *// Add the last digit of n to sum*

*n* /= 10; *// Remove the last digit from n*

    }

    return sum; *// Return the computed sum of digits*

}

*// Test example*

int main() {

    int num;

*// Input number for which sum of digits is to be calculated*

    std::cout << "Enter a number: ";

    std::cin >> num;

*// Calculate sum of digits*

    int result = sumOfDigits(num);

*// Output the sum of digits*

    std::cout << "Sum of digits of " << num << " is: " << result << std::endl;

    return 0; *// Exit the program*

}

**Output**

Enter a number: 45896587

Sum of digits of 45896587 is: 52

1. 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>

*// Function to calculate the GCD of two numbers*

int gcd(int *a*, int *b*) {

*// Euclidean algorithm to find GCD*

    while (*b* != 0) {

        int temp = *b*;

*b* = *a* % *b*;

*a* = temp;

    }

    return *a*; *// Return the GCD*

}

*// Test example*

int main() {

    int num1, num2;

*// Input two numbers for which GCD is to be calculated*

    std::cout << "Enter first number: ";

    std::cin >> num1;

    std::cout << "Enter second number: ";

    std::cin >> num2;

*// Calculate GCD of num1 and num2*

    int result = gcd(num1, num2);

*// Output the GCD*

    std::cout << "GCD of " << num1 << " and " << num2 << " is: " << result << std::endl;

    return 0; *// Exit the program*

}

**Output**

Enter first number: 545

Enter second number: 658

GCD of 545 and 658 is: 1

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.

#include <iostream>

#include <vector>

#include <algorithm> *// For std::min\_element and std::max\_element*

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

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

*// Check if the array has at least two elements*

    if (nums.size() < 2) {

        return 0; *// If less than two elements, return 0 (no valid difference)*

    }

*// Find the minimum and maximum elements in the array*

    int minElement = \*std::min\_element(nums.begin(), nums.end());

    int maxElement = \*std::max\_element(nums.begin(), nums.end());

*// Calculate and return the maximum difference*

    return maxElement - minElement;

}

*// Test example*

int main() {

    std::vector<int> nums;

    int n;

*// Input size of array*

    std::cout << "Enter number of elements: ";

    std::cin >> n;

*// Input array elements*

    std::cout << "Enter elements of array: ";

    for (int i = 0; i < n; ++i) {

        int num;

        std::cin >> num;

        nums.push\_back(num);

    }

*// Calculate maximum difference*

    int result = maxDifference(nums);

*// Output the maximum difference*

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

    return 0; *// Exit the program*

}

**Output**

Enter number of elements: 5

Enter elements of array: 1 9 3 7 5

Maximum difference between any two elements in the array is: 8

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.

#include <iostream>

#include <cctype> *// For std::isalpha*

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

bool containsOnlyAlphabetic(const std::string& *str*) {

*// Iterate through each character in the string*

    for (char c : *str*) {

*// Check if the character is not alphabetic*

        if (!std::isalpha(c)) {

            return false; *// Return false if non-alphabetic character found*

        }

    }

    return true; *// Return true if all characters are alphabetic*

}

*// Test example*

int main() {

    std::string input;

*// Input string to check*

    std::cout << "Enter a string: ";

    std::getline(std::cin, input);

*// Check if the string contains only alphabetic characters*

    bool result = containsOnlyAlphabetic(input);

*// Output the result*

    std::cout << std::boolalpha; *// Print bool values as true/false*

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

    return 0; *// Exit the program*

}

**Output**

Enter a string: zesxdrcftvgh23456

The string contains only alphabetic characters: false