**Global Trend Programming Profile 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.**

#include <iostream>

Using namespace std;

struct ListNode {

int val;

ListNode\* next;

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

};

ListNode\* reverseLinkedList(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\* head) {

ListNode\* temp = head;

while (temp != nullptr) {

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

temp = temp->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);

cout << "Original list: ";

printList(head);

ListNode\* newHead = reverseLinkedList(head);

cout << "Reversed list: ";

printList(newHead);

return 0;

}

**OUTPUT**

**A black screen with white text

Description automatically generated**

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\_set>

#include <algorithm>

Using namespace std;

int longestSubstring(string s) {

int n = s.length();

int maxLen = 0, i = 0, j = 0;

unordered\_set<char> charSet;

while (j < n) {

if (charSet.find(s[j]) == charSet.end()) {

charSet.insert(s[j++]);

maxLen = std::max(maxLen, j - i);

} else {

charSet.erase(s[i++]);

}

}

return maxLen;

}

int main() {

string input;

cout << "Enter a string: ";

cin >> input;

int result = longestSubstring(input);

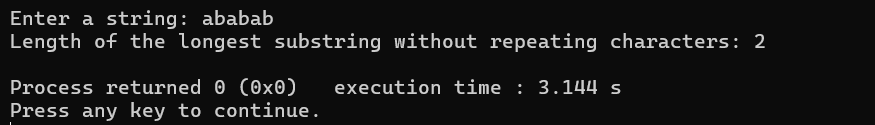
cout << "Length of the longest substring without repeating characters: " << result << endl;

return 0;

}

**OUTPUT**

**A black background with white text

Description automatically generated**

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>

Using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

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

};

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

if (node == nullptr) return 0;

int leftSum = max(0, maxPathSumHelper(node->left, maxSum));

int rightSum = max(0, maxPathSumHelper(node->right, maxSum));

int currentSum = node->val + leftSum + rightSum;

maxSum = max(maxSum, currentSum);

return node->val + max(leftSum, rightSum);

}

int maxPathSum(TreeNode\* root) {

int maxSum = INT\_MIN;

maxPathSumHelper(root, maxSum);

return maxSum;

}

TreeNode\* createTree() {

int val;

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

cin >> val;

if (val == -1) {

return nullptr;

}

TreeNode\* node = new TreeNode(val);

cout << "Enter left child of " << val << ":" << endl;

node->left = createTree();

cout << "Enter right child of " << val << ":" << endl;

node->right = createTree();

return node;

}

int main() {

cout << "Enter nodes of the binary tree (enter -1 for null):" << endl;

TreeNode\* root = createTree();

int result = maxPathSum(root);

cout << "Maximum path sum in the binary tree: " << result << endl;

return 0;

}

**OUTPUT**

A computer screen shot of a black screen

Description automatically generated

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>

#include <string>

#include <queue>

using namespace std;

struct TreeNode {

int val;

TreeNode \*left;

TreeNode \*right;

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

};

class Codec {

public:

string serialize(TreeNode\* root) {

if (!root) return "X,";

return to\_string(root->val) + "," + serialize(root->left) + serialize(root->right);

}

TreeNode\* deserialize(string data) {

stringstream ss(data);

return deserializeHelper(ss);

}

private:

TreeNode\* deserializeHelper(stringstream& ss) {

string val;

getline(ss, val, ',');

if (val == "X") return NULL;

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

node->left = deserializeHelper(ss);

node->right = deserializeHelper(ss);

return node;

}

};

int main() {

Codec codec;

int numNodes;

cout << "Enter the number of nodes in the binary tree: ";

cin >> numNodes;

TreeNode\* root = NULL;

queue<TreeNode\*\*> q;

for (int i = 0; i < numNodes; i++) {

int val;

cout << "Enter the value of node " << i + 1 << ": ";

cin >> val;

TreeNode\* node = new TreeNode(val);

if (!root) {

root = node;

q.push(&root);

} else {

TreeNode\*\* front = q.front();

q.pop();

if (!\*front) {

q.push(&(\*front));

q.push(&node);

} else {

if (!(\*front)->left) {

(\*front)->left = node;

q.push(&((\*front)->left));

} else {

(\*front)->right = node;

q.push(&((\*front)->right));

}

}

}

}

string serialized = codec.serialize(root);

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

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

queue<TreeNode\*> q2;

q2.push(deserializedRoot);

cout << "Deserialized: ";

while (!q2.empty()) {

TreeNode\* node = q2.front();

q2.pop();

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

if (node->left) q2.push(node->left);

if (node->right) q2.push(node->right);

}

cout << endl;

return 0;

}

**OUTPUT**

A black screen with white text

Description automatically generated

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>

Using namespace std;

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

int n = nums.size();

k = k % n;

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

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

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

}

int main() {

int n, k;

cout << "Enter number of elements in the array: ";

cin >> n;

vector<int> nums(n);

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

for (int i = 0; i < n; ++i) {

cin >> nums[i];

}

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

cin >> k;

rotate(nums, k);

cout << "Array after rotating right by " << k << " steps: ";

for (int num : nums) {

cout << num << " ";

}

cout << endl;

return 0;

}

**OUTPUT**

A black screen with white text

Description automatically generated

1. **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 number) {

if (number <= 1) {

return 1;

}

return number \* factorial(number - 1);

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (num < 0) {

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

return 1;

}

int result = factorial(num);

cout << "The factorial of " << num << " is: " << result << endl;

return 0;

}

**OUTPUT**

A black screen with white text

Description automatically generated

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>

Using namespace std;

int sumOfDigits(int number) {

int sum = 0;

while (number != 0) {

sum += number % 10;

number /= 10;

}

return sum;

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

int result = sumOfDigits(num);

cout << "The sum of the digits of " << num << " is: " << result << endl;

return 0;

}

**OUTPUT**

A black background with white text

Description automatically generated

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>

Using namespace std;

int findGCD(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int main() {

int num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

int gcd = findGCD(num1, num2);

cout << "The greatest common divisor (GCD) of " << num1 << " and " << num2 << " is: " << gcd << endl;

return 0;

}

**OUTPUT**

A screen shot of a computer

Description automatically generated

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>

using namespace std;

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

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

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

return maxElement - minElement;

}

int main() {

int n;

cout << "Enter the number of elements in the array: ";

cin >> n;

if (n < 2) {

cerr << "The array should have at least two elements." << endl;

return 1;

}

vector<int> array(n);

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

for (int i = 0; i < n; ++i) {

cin >> array[i];

}

int maxDifference = findMaxDifference(array);

cout << "The maximum difference between any two elements is: " << maxDifference << endl;

return 0;

}

**OUTPUT**

A black screen with white text

Description automatically generated

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 <string>

#include <cctype>

using namespace std;

bool isAlphabetic(const string& str) {

for (char ch : str) {

if (!isalpha(ch)) { >

return false;

}

}

return true;

}

int main() {

string userInput;

cout << "Enter a string: ";

getline(cin, userInput);

cout << "The string contains only alphabetic characters: ";

cout << (isAlphabetic(userInput) ? "True" : "False") << endl;

return 0;

}

**OUTPUT**

A black background with white text

Description automatically generated A black screen with white text

Description automatically generated