Q1

Code:

#include <iostream>

// Definition for singly-linked list.

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;

}

// Function to print linked list

void printList(ListNode\* node) {

while (node != nullptr) {

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

node = node->next;

}

}

int main() {

// Create a linked list 1->2->3->4->5

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);

std::cout << std::endl;

ListNode\* reversedHead = reverseList(head);

std::cout << "Reversed List: ";

printList(reversedHead);

std::cout << std::endl;

return 0;

}

Output:

Original List: 1 2 3 4 5

Reversed List: 5 4 3 2 1

Q2

Code:

#include <iostream>

#include <string>

#include <unordered\_map>

#include <algorithm>

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

std::unordered\_map<char, int> charMap;

int maxLength = 0, start = 0;

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

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

start = std::max(start, charMap[s[end]] + 1);

}

charMap[s[end]] = end;

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

}

return maxLength;

}

int main() {

std::string s = "abcabcbb";

std::cout << "Length of Longest Substring Without Repeating Characters: " << lengthOfLongestSubstring(s) << std::endl;

return 0;

}

Output:

Length of Longest Substring Without Repeating Characters: 3

Q3

Code:

#include <iostream>

#include <algorithm>

// Definition for a binary tree node.

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

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

};

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

if (node == nullptr) return 0;

int leftSum = std::max(0, maxPathSumUtil(node->left, maxSum));

int rightSum = std::max(0, maxPathSumUtil(node->right, maxSum));

maxSum = std::max(maxSum, node->val + leftSum + rightSum);

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

}

int maxPathSum(TreeNode\* root) {

int maxSum = INT\_MIN;

maxPathSumUtil(root, maxSum);

return maxSum;

}

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);

std::cout << "Maximum Path Sum: " << maxPathSum(root) << std::endl;

return 0;

}

Output:

Maximum Path Sum: 42

Q4

Code:

#include <iostream>

#include <string>

#include <sstream>

#include <queue>

// 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.

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,";

}

}

std::string s = ss.str();

s.pop\_back(); // Remove trailing comma

return s;

}

// Decodes your encoded data to tree.

TreeNode\* deserialize(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;

}

};

int main() {

Codec ser, deser;

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);

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

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

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

std::cout << "Deserialized Tree (root value): " << deserialized->val << std::endl;

return 0;

}

Output:

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

Deserialized Tree (root value): 1

Q5

Code:

#include <iostream>

#include <vector>

#include <algorithm>

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

k %= nums.size();

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

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

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

}

int main() {

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

int k = 3;

rotate(nums, k);

std::cout << "Rotated Array: ";

for (int num : nums) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Output:

Rotated Array: 5 6 7 1 2 3 4

Q6

Code:

#include <iostream>

int factorial(int n) {

if (n == 0) return 1;

return n \* factorial(n - 1);

}

int main() {

int n = 5;

std::cout << "Factorial of " << n << " is: " << factorial(n) << std::endl;

return 0;

}

Output:

Factorial of 5 is: 120

Q7

#include <iostream>

int sumOfDigits(int n) {

int sum = 0;

while (n > 0) {

sum += n % 10;

n /= 10;

}

return sum;

}

int main() {

int n = 12345;

std::cout << "Sum of digits of " << n << " is: " << sumOfDigits(n) << std::endl;

return 0;

}

Output:

Sum of digits of 12345 is: 15

Q8

Code:

#include <iostream>

int gcd(int a, int b) {

if (b == 0) return a;

return gcd(b, a % b);

}

int main() {

int a = 56, b = 98;

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

return 0;

}

Output:

GCD of 56 and 98 is: 14

Q9

Code:

#include <iostream>

#include <vector>

#include <algorithm>

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

if (nums.empty()) return 0;

int minElement = nums[0];

int maxDiff = 0;

for (int i = 1; i < nums.size(); ++i) {

maxDiff = std::max(maxDiff, nums[i] - minElement);

minElement = std::min(minElement, nums[i]);

}

return maxDiff;

}

int main() {

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

std::cout << "Maximum difference: " << maxDifference(nums) << std::endl;

return 0;

}

Output:

Maximum difference: 5

Q10

Code:

#include <iostream>

#include <string>

#include <cctype>

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

for (char c : s) {

if (!std::isalpha(c)) {

return false;

}

}

return true;

}

int main() {

std::string s = "HelloWorld";

std::cout << "Is the string alphabetic? " << (isAlphabetic(s) ? "Yes" : "No") << std::endl;

return 0;

}

Output:

Is the string alphabetic? Yes