Q1) 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 1)

#include <iostream>

using namespace std;

// Definition for singly-linked list.

struct ListNode {

    int val;

    ListNode\* next;

// constructor

    ListNode(int x){

      this->val = x;

      this->next = NULL;

}

};

// Function to reverse the singly linked list

ListNode\* reverseList(ListNode\* head) {

    ListNode\* prev = NULL;     // Previous node, initially set to NULL

    ListNode\* curr = head;     // Current node, initially set to head

    ListNode\* next = NULL;     // Next node, initially set to NULL

    while (curr != NULL) {

        next = curr->next;     // Store the next node

        curr->next = prev;     // Reverse the current node's pointer

        prev = curr;           // Move the previous node one step forward

        curr = next;           // Move the current node one step forward

    }

    return prev;               // New head of the reversed list

}

// Function to print the linked list

void printList(ListNode\* head) {

    ListNode\* temp = head;

    while (temp != NULL) {

        cout << temp->val << " ";

        temp = temp->next;

    }

    cout << endl;

}

// Helper function to insert a new node at the end of the list

void insert(ListNode\*& head, int val) {

    if (head == NULL) {

        head = new ListNode(val);

    } else {

        ListNode\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = new ListNode(val);

    }

}

int main() {

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

    ListNode\* head = NULL;

    insert(head, 1);

    insert(head, 2);

    insert(head, 3);

    insert(head, 4);

    insert(head, 5);

    cout << "Original list: ";

    printList(head);

    // Reverse the linked list

    head = reverseList(head);

    cout << "Reversed list: ";

    printList(head);

    return 0;

}

// output

// Original list: 1 2 3 4 5

// Reversed list: 5 4 3 2 1

Q2) 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 2)

#include <iostream>

#include <unordered\_map>

#include <algorithm>

using namespace std;

int lengthOfLongestSubstring(string s) {

   // To store the most recent index of each character

    unordered\_map<char, int> charIndexMap;

   // To store the maximum length of substring without repeating characters

    int maxLength = 0;

// Start index of the current window

    int start = 0;

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

        char currentChar = s[end];

        // If the character is found in the map and is within the current window

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

            // Move the start to the right of the duplicate character's last known position

            start = charIndexMap[currentChar] + 1;

        }

        // Update the most recent index of the current character

        charIndexMap[currentChar] = end;

        // Calculate the length of the current window and update maxLength if necessary

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

    }

    return maxLength;

}

int main() {

    string s = "abcabcbb";

    cout << "The length of the longest substring without repeating characters is: " << lengthOfLongestSubstring(s) << endl;

    s = "bbbbb";

    cout << "The length of the longest substring without repeating characters is: " << lengthOfLongestSubstring(s) << endl;

    s = "pwwkew";

    cout << "The length of the longest substring without repeating characters is: " << lengthOfLongestSubstring(s) << endl;

    s = "aab";

    cout << "The length of the longest substring without repeating characters is: " << lengthOfLongestSubstring(s) << endl;

    s = "dvdf";

    cout << "The length of the longest substring without repeating characters is: " << lengthOfLongestSubstring(s) << endl;

    return 0;

}

// output

// The length of the longest substring without repeating characters is: 3

// The length of the longest substring without repeating characters is: 1

// The length of the longest substring without repeating characters is: 3

// The length of the longest substring without repeating characters is: 2

// The length of the longest substring without repeating characters is: 3

Q3) 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 3)

#include <iostream>

#include <algorithm>

#include <climits>

using namespace std;

// Definition for a binary tree node.

struct TreeNode {

    int val;

    TreeNode \*left;

    TreeNode \*right;

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

};

class Solution {

public:

    int maxPathSum(TreeNode\* root) {

        int maxSum = INT\_MIN;

        maxGain(root, maxSum);

        return maxSum;

    }

private:

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

        if (node == NULL) {

            return 0;

        }

        // Recursively get the maximum gain from left and right subtrees

        int leftGain = max(maxGain(node->left, maxSum), 0); // Ignore paths with negative sum

        int rightGain = max(maxGain(node->right, maxSum), 0);

        // The price to start a new path where `node` is the highest node

        int newPathSum = node->val + leftGain + rightGain;

        // Update maxSum if it's better to start a new path

        maxSum = max(maxSum, newPathSum);

        // For recursion, return the maximum gain if continuing the same path

        return node->val + max(leftGain, rightGain);

    }

};

// Helper function to create a new tree node

TreeNode\* newNode(int val) {

    TreeNode\* node = new TreeNode(val);

    node->left = NULL;

    node->right = NULL;

    return node;

}

int main() {

    // Construct the tree

    TreeNode\* root = newNode(-10);

    root->left = newNode(9);

    root->right = newNode(20);

    root->right->left = newNode(15);

    root->right->right = newNode(7);

    Solution solution;

    cout << "The maximum path sum is: " << solution.maxPathSum(root) << endl;

    return 0;

}

// output

// The maximum path sum is: 42

Q4) 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 4)

#include <iostream>

#include <sstream>

#include <string>

#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(NULL), right(NULL) {}

};

class Codec {

public:

    // Encodes a tree to a single string.

    string serialize(TreeNode\* root) {

        if (!root) return "";

        ostringstream out;

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            TreeNode\* node = q.front();

            q.pop();

            if (node) {

                out << node->val << " ";

                q.push(node->left);

                q.push(node->right);

            } else {

                out << "# ";

            }

        }

        return out.str();

    }

    // Decodes your encoded data to tree.

    TreeNode\* deserialize(string data) {

        if (data.empty()) return nullptr;

        istringstream in(data);

        string val;

        in >> val;

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

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            TreeNode\* node = q.front();

            q.pop();

            if (!(in >> val)) break;

            if (val != "#") {

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

                q.push(node->left);

            }

            if (!(in >> val)) break;

            if (val != "#") {

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

                q.push(node->right);

            }

        }

        return root;

    }

};

// Helper function to print a tree (in-order traversal)

void printTree(TreeNode\* root) {

    if (!root) return;

    printTree(root->left);

    cout << root->val << " ";

    printTree(root->right);

}

int main() {

    // Construct a sample 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 codec;

    string serialized = codec.serialize(root);

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

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

    cout << "Deserialized tree (in-order traversal): ";

    printTree(deserializedRoot);

    cout << endl;

    return 0;

}

// output

// Serialized tree: 1 2 3 # # 4 5 # # # #

// Deserialized tree (in-order traversal): 2 1 4 3 5

Q5) 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 5)

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Helper function to reverse a portion of the array

void reverse(vector<int>& nums, int start, int end) {

    while (start < end) {

        swap(nums[start], nums[end]);

        start++;

        end--;

    }

}

// Function to rotate the array to the right by k steps

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

    int n = nums.size();

    k = k % n;  // Handle cases where k is greater than the array size

    // Step 1: Reverse the entire array

    reverse(nums, 0, n - 1);

    // Step 2: Reverse the first k elements

    reverse(nums, 0, k - 1);

    // Step 3: Reverse the remaining n-k elements

    reverse(nums, k, n - 1);

}

int main() {

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

    int k = 3;

    cout << "Original array: ";

    for (int num : nums) {

        cout << num << " ";

    }

    cout << endl;

    rotate(nums, k);

    cout << "Rotated array: ";

    for (int num : nums) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}

// output

// Original array: 1 2 3 4 5 6 7

// Rotated array: 5 6 7 1 2 3 4

Q6) Write a function to find the factorial of a given number.The function should return the factorial of the number.

Code 6)

#include <iostream>

using namespace std;

// Function to calculate the factorial of a number

unsigned long long factorial(int n) {

    if (n < 0) {

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

        return 0;

    }

    unsigned long long result = 1;

    for (int i = 2; i <= n; ++i) {

        result \*= i;

    }

    return result;

}

int main() {

    int num;

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

    cin >> num;

    unsigned long long fact = factorial(num);

    cout << "Factorial of " << num << " is: " << fact << endl;

    return 0;

}

// output

// Enter a number to find its factorial: 5

// Factorial of 5 is: 120

Q7) 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 7)

#include <iostream>

using namespace std;

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

int sumOfDigits(int num) {

    // Handle negative numbers by taking the absolute value

    num = abs(num);

    int sum = 0;

    while (num > 0) {

        sum += num % 10; // Add the last digit to the sum

        num /= 10;       // Remove the last digit

    }

    return sum;

}

int main() {

    int num;

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

    cin >> num;

    int sum = sumOfDigits(num);

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

    return 0;

}

// output

// Enter a number to compute the sum of its digits: 123

// Sum of the digits of 123 is: 6

Q8) Write a function to find the greatest common divisor (GCD) of two numbers. The function should return the GCD of a and b.

Code 8)

#include <iostream>

using namespace std;

// Function to compute the GCD of two numbers using the Euclidean algorithm

int gcd(int a, int b) {

    // Handle the case where one of the numbers is zero

    if (b == 0) {

        return a;

    }

    // Recursive call with parameters (b, a % b)

    return gcd(b, a % b);

}

int main() {

    int a, b;

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

    cin >> a >> b;

    int result = gcd(a, b);

    cout << "The GCD of " << a << " and " << b << " is: " << result << endl;

    return 0;

}

// output

// Enter two numbers to find their GCD: 10 2

// The GCD of 10 and 2 is: 2

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.

Code 9)

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

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

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

    if (nums.empty()) {

        cerr << "Array is empty. Cannot compute difference." << endl;

        return 0;

    }

    int minElement = nums[0];

    int maxElement = nums[0];

    // Traverse the array to find the minimum and maximum elements

    for (int num : nums) {

        if (num < minElement) {

            minElement = num;

        }

        if (num > maxElement) {

            maxElement = num;

        }

    }

    // The maximum difference is between the maximum and minimum elements

    return maxElement - minElement;

}

int main() {

    vector<int> nums = {2, 9, 5, 1, 7, 3};

    int maxDiff = maxDifference(nums);

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

    return 0;

}

//  output

// The maximum difference between any two elements is: 8

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.

Code 10)

#include <iostream>

#include <cctype> // For isalpha function

#include <string>

using namespace std;

// Function to check if the string contains only alphabetic characters

bool isAlphabetic(const string& str) {

    for (char ch : str) {

        if (!isalpha(ch)) {

            return false;

        }

    }

    return true;

}

int main() {

    string input;

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

    cin >> input;

    if (isAlphabetic(input)) {

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

    } else {

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

    }

    return 0;

}

// output

// Enter a string to check if it contains only alphabetic characters: hello world

// The string contains only alphabetic characters.

// Enter a string to check if it contains only alphabetic characters: hwll0

// The string contains non-alphabetic characters.