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.

// creating the linked list

#include <iostream>

using namespace std;

class ListNode{

public:

int data;

ListNode\* next;

ListNode\* head;

ListNode(int data){

this->data=data;

this->next=NULL;

}

};

// function to create the linked list

void addIntoList(ListNode\* &head, int data){

ListNode\* temp= new ListNode(data);

if(head==NULL){

head=temp;

return ;

}

ListNode\* curr=head;

while(curr->next){

curr=curr->next;

}

curr->next=temp;

return;

}

// function to print the linked list

void printList(ListNode\* head){

ListNode\* temp=head;

if(temp==NULL)

cout<<"list is empty";

while(temp){

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

}

// function to reverse the linked list

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

if(head==NULL || head->next == NULL){

return head;

}

ListNode\* newHead= reverseList(head->next);

head->next->next=head;

head->next=NULL;

return newHead;

}

// main function

int main(){

ListNode\* head=NULL;

addIntoList(head,9);

addIntoList(head,5);

addIntoList(head,6);

addIntoList(head,9);

addIntoList(head,11);

addIntoList(head,3);

//printing list before reverse;

printList(head);

ListNode\* newHead = reverseList(head);

head=newHead;

cout<<"new head of reversed linked list: "<<head->data<<endl;

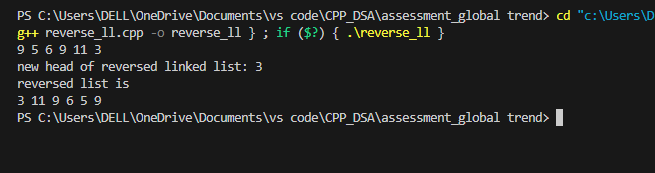
//printing list after reverse;

cout<<"reversed list is"<<endl;

printList(head);

}

OUTPUT



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>

using namespace std;

int longestSubstr(string s){

unordered\_map<char,int> mp;

int maxlen=0;

int left=0;

int right=0;

while(right<s.length()){

if(mp.find(s[right])!=mp.end() && mp[s[right]]>=left){

left=mp[s[right]]+1;

}

maxlen=max(maxlen,right-left+1);

mp[s[right]]=right;

right++;

}

return maxlen;

}

int main(){

cout<<longestSubstr("aaaaa")<<endl;

cout<<longestSubstr("abcdefgh")<<endl;

cout<<longestSubstr("ababab")<<endl;

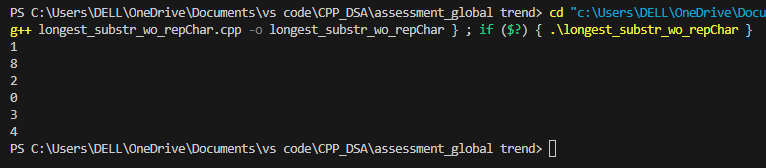
cout<<longestSubstr("")<<endl;

cout<<longestSubstr("aabcaa")<<endl;

cout<<longestSubstr("abcddd")<<endl;

}

OUTPUT



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 <bits/stdc++.h>

#include <queue>

using namespace std;

class TreeNode{

public:

int data;

TreeNode\* right;

TreeNode\* left;

TreeNode(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

// creating tree using level order traversal

TreeNode\* buildTree(TreeNode\* &root){

cout<<"enter the root data"<<endl;

int data;

cin>>data;

root =new TreeNode(data);

queue<TreeNode\*> q;

q.push(root);

while(!q.empty()){

TreeNode\* front=q.front();

q.pop();

cout<<"enter the left of: "<<front->data<<endl;

int leftData;

cin>>leftData;

if(leftData!=-1){

front->left=new TreeNode(leftData);

q.push(front->left);

}

cout<<"enter the right of: "<<front->data<<endl;

int rightData;

cin>>rightData;

if(rightData!=-1){

front->right=new TreeNode(rightData);

q.push(front->right);

}

}

return root;

}

void printLevelOrder(TreeNode\* root){

queue<TreeNode\*> q;

q.push(root);

q.push(NULL);

while(!q.empty()){

TreeNode\* front= q.front();

q.pop();

if(front==NULL){

cout<<endl;

if(!q.empty()) {

q.push(NULL);

}

continue;

}

cout<<front->data<<" ";

if(front->left)

q.push(front->left);

if(front->right)

q.push(front->right);

}

}

// -10 5 6 7 8 9 -1 11 -1 -1 -1 -1 -1 -1 -1

int maxPathSum(TreeNode\* &root, int& sum){

if(root==NULL){

return 0;

}

int leftSum=maxPathSum(root->left,sum);

int rightSum=maxPathSum(root->right,sum);

sum=max(sum,root->data+rightSum+leftSum);

return max(0,root->data+max(leftSum,rightSum));

}

int main(){

TreeNode\* root = NULL;

root = buildTree(root);

//printing the list level order

printLevelOrder(root);

//finding maximum path sum

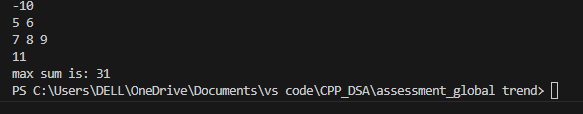
int sum=INT\_MIN;

maxPathSum(root,sum);

cout<<"max sum is: "<<sum<<endl;

}

Output



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 <queue>

#include<string>

#include <bits/stdc++.h>

using namespace std;

class TreeNode{

public:

int data;

TreeNode\* right;

TreeNode\* left;

TreeNode(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

// creating tree using level order traversal

TreeNode\* buildTree(TreeNode\* &root){

cout<<"enter the root data"<<endl;

int data;

cin>>data;

root =new TreeNode(data);

queue<TreeNode\*> q;

q.push(root);

while(!q.empty()){

TreeNode\* front=q.front();

q.pop();

cout<<"enter the left of: "<<front->data<<endl;

int leftData;

cin>>leftData;

if(leftData!=-1){

front->left=new TreeNode(leftData);

q.push(front->left);

}

cout<<"enter the right of: "<<front->data<<endl;

int rightData;

cin>>rightData;

if(rightData!=-1){

front->right=new TreeNode(rightData);

q.push(front->right);

}

}

return root;

}

void printLevelOrder(TreeNode\* root){

queue<TreeNode\*> q;

q.push(root);

q.push(NULL);

while(!q.empty()){

TreeNode\* front= q.front();

q.pop();

if(front==NULL){

cout<<endl;

if(!q.empty()) {

q.push(NULL);

}

continue;

}

cout<<front->data<<" ";

if(front->left)

q.push(front->left);

if(front->right)

q.push(front->right);

}

}

string ser(TreeNode\* &root){

if(root==NULL) return "";

string s="";

queue<TreeNode\*> q;

q.push(root);

while(!q.empty()){

TreeNode\* front=q.front();

q.pop();

if(front==NULL){

s=s+"#,";

}

else{

s=s+to\_string(front->data)+',';

}

if(front){

q.push(front->left);

q.push(front->right);

}

}

return s;

}

TreeNode\* deSer(string data){

if (data.size()==0) {

return NULL;

}

stringstream s(data);

string str;

getline(s, str, ',');

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

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

getline(s, str, ',');

if (str != "#") {

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

node->left = leftNode;

q.push(leftNode);

}

getline(s, str, ',');

if (str != "#") {

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

node->right = rightNode;

q.push(rightNode);

}

}

return root;

}

// -10 5 6 7 8 9 -1 11 -1 -1 -1 -1 -1 -1 -1

int main(){

TreeNode\* root=NULL;

root= buildTree(root);

printLevelOrder(root);

string s;

s= ser(root);

cout<<s<<endl;

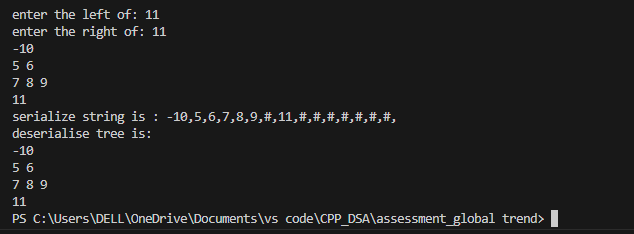
TreeNode\* newRoot=NULL;

newRoot=deSer(s);

printLevelOrder(newRoot);

}

OUTPUT



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>

using namespace std;

// reversing funcion

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

while (start < end) {

swap(nums[start], nums[end]);

start++;

end--;

}

}

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

int n = nums.size();

k = k % n;

if (k == 0) return;

// Reverse the entire array

reverse(nums, 0, n-1);

// Reverse the first k elements

reverse(nums, 0, k-1);

// Reverse the remaining elements

reverse(nums, k, n-1);

}

void printArray(vector<int> nums){

for (int i=0;i<nums.size();i++) {

cout << nums[i] << " ";

}

cout << endl;

}

int main() {

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

int k = 3;

cout << "Original array: ";

printArray(nums);

rotate(nums, k);

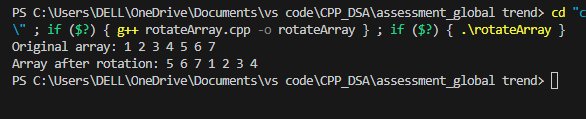
cout << "Array after rotation: ";

printArray(nums);

return 0;

}

OUTPUT



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;

unsigned long long fact(int n) {

if (n <= 1) return 1;

return n \* fact(n - 1);

}

int main() {

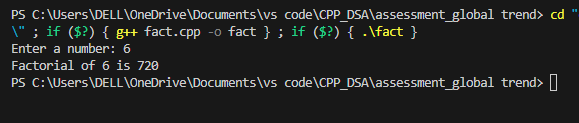
int number;

cout << "Enter a number: ";

cin >> number;

cout << "Factorial of " << number << " is " << fact(number) << endl;

}

OUTPUT

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 n) {

int sum = 0;

n = abs(n);

while (n > 0) {

sum += n % 10;

n /= 10;

}

return sum;

}

int main() {

int n;

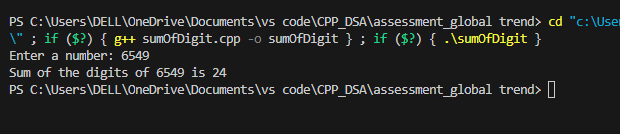
cout << "Enter a number: ";

cin >> n;

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

}

OUTPUT



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 gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int main() {

int num1, num2;

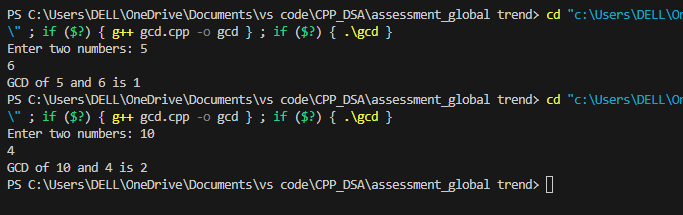
cout << "Enter two numbers: ";

cin >> num1 >> num2;

cout << "GCD of " << num1 << " and " << num2 << " is " << gcd(num1, num2) << endl;

}

ouput



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>

using namespace std;

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

if (nums.empty()) {

return -1;

}

int minElement = nums[0];

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

if (nums[i] < minElement) {

minElement = nums[i];

}

}

return minElement;

}

// Function to find the max difference

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

if (nums.empty() || nums.size()==1) return -1;

int minElement = findMinElement(nums); // Find the minimum element in the array

int maxDiff = 0;

for (int i = 0; i < nums.size(); ++i) {

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

}

return maxDiff;

}

int main() {

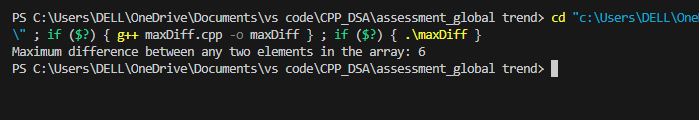
vector<int> nums = {7, 1, 5, 3, 6, 4};

cout << "Maximum difference between any two elements in the array: "

<< maxDifference(nums) << endl;

}

OUTPUT



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>

using namespace std;

bool containsOnlyAlphabets(const string& str) {

for (char c : str) {

if (!isalpha(c)) {

return false;

}

}

return true;

}

int main() {

string input;

cout << "Enter a string: ";

getline(cin, input);

if (containsOnlyAlphabets(input)) {

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

} else {

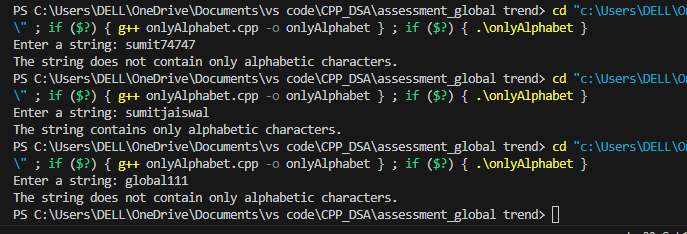
cout << "The string does not contain only alphabetic characters." << endl;

}

return 0;

}

Output



Thank you

-----------------------------------------------------------------------