**SOLUTION 1**

#include<iostream>

using namespace std;

class Node {

public:

int value;

Node \*next;

//construcutor

Node(int val){

this->value=val;

this->next=NULL;

}

};

void insertattail(Node\*&tail,int f){

Node\*Node2=new Node(f);

tail->next=Node2;

tail=Node2;

}

void insertathead(Node\* &head,int d){

Node \*temp = new Node(d);

temp->next=head;

head=temp;

}

void print(Node \* &head){

Node\*tem=head;

while(tem!=NULL){

cout<< tem->value<<" ";

tem=tem->next;

}

cout<<endl;

}

Node\* reverseList(Node\*head) {

Node\*prev=NULL;

Node\*curr=head;

Node\*forward=NULL;

if(head==NULL){

return{};

}

if(head->next==NULL){

return head;

}

while(curr!=NULL){

forward=curr->next;

curr->next=prev;

prev=curr;

curr=forward;

}

return prev;

}

int main(){

Node\*Node1= new Node(10);

Node\*head=Node1;

Node\*tail=Node1;

insertattail(tail,11);

insertattail(tail,12);

insertattail(tail,13);

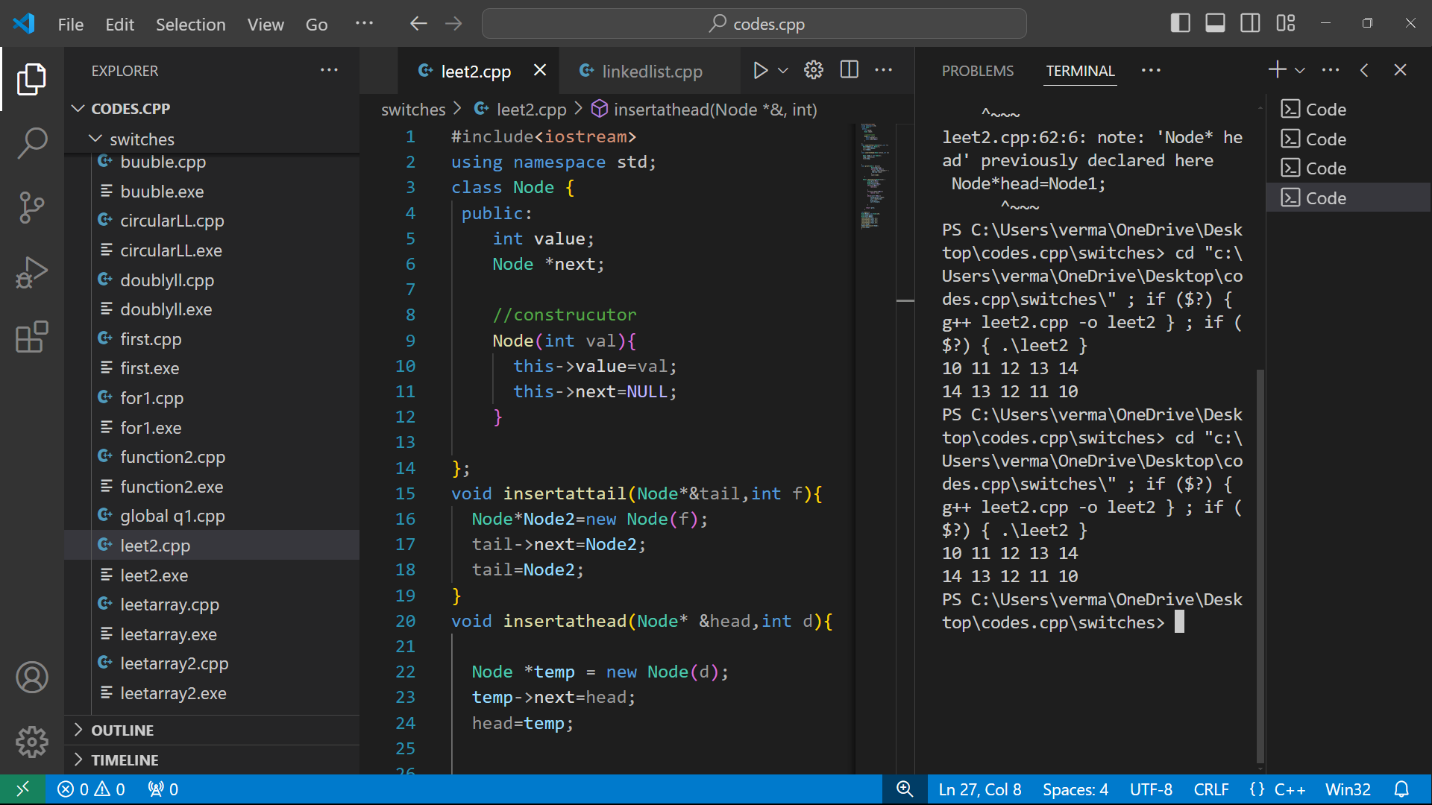
insertattail(tail,14);

print(head);

head=reverseList(head);

print(head);

}



**SOLUTION 2**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

int length(string s) {

int n = s.length();

int maxLength = 0;

vector<int> charIndex(128, -1);

int left = 0;

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

if (charIndex[s[right]] >= left) {

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

}

charIndex[s[right]] = right;

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

}

return maxLength;

}

int main() {

string s = "abcabcbb";

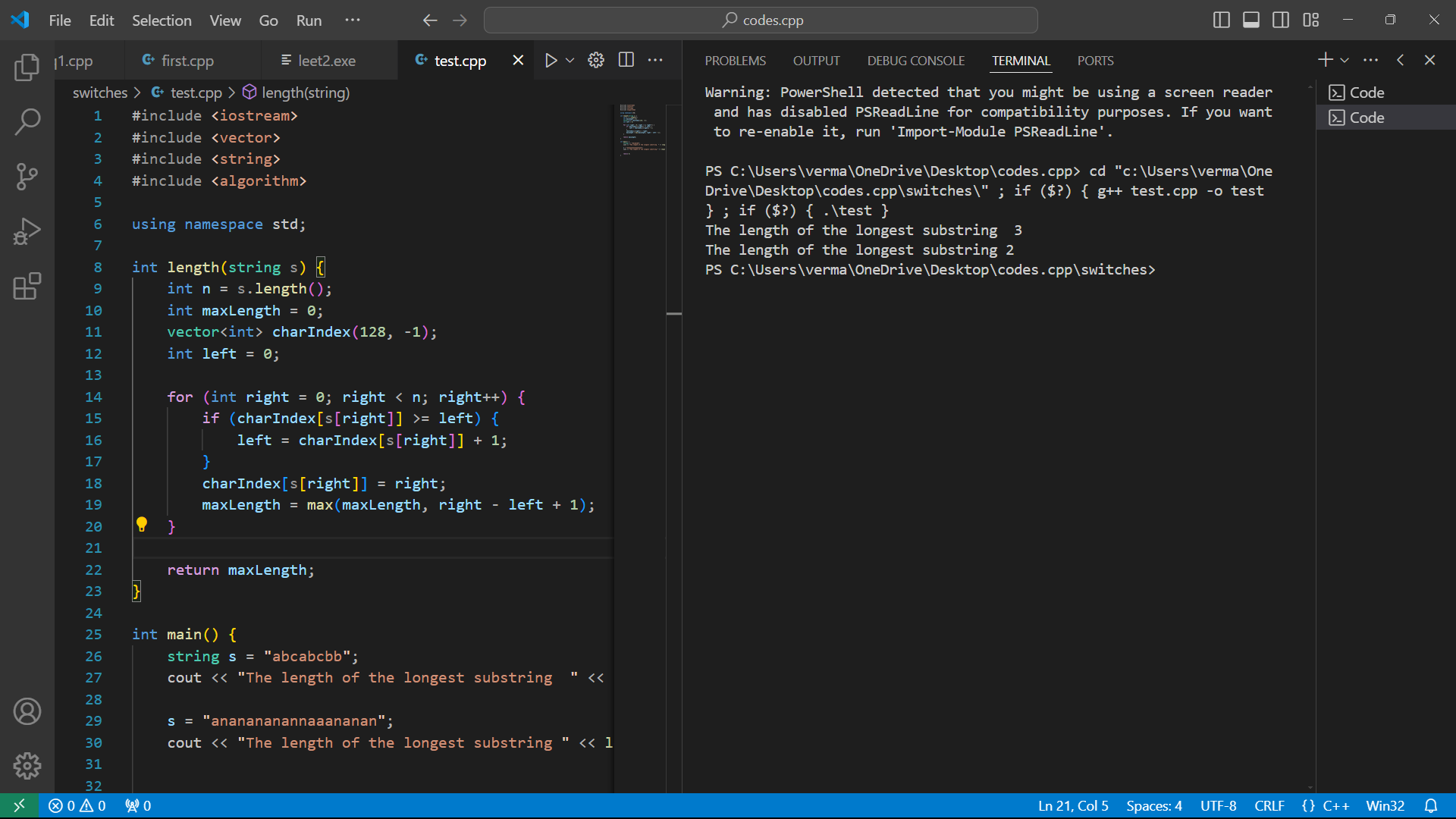
cout << "The length of the longest substring " << length(s) << endl;

s = "ananananannaaananan";

cout << "The length of the longest substring " << length(s) << endl;

return 0;

}



**SOLUTION 3**

#include <iostream>

#include <climits>

using namespace std;

class TreeNode {

public:

int val;

TreeNode\* left;

TreeNode\* right;

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

};

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 == nullptr) {

return 0;

}

int leftGain = max(maxGain(node->left, maxSum), 0);

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

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

maxSum = max(maxSum, currentSum);

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

}

};

int main() {

TreeNode\* root = new TreeNode(1);

root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(5);

Solution sol;

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

// Clean up memory

delete root->left->right;

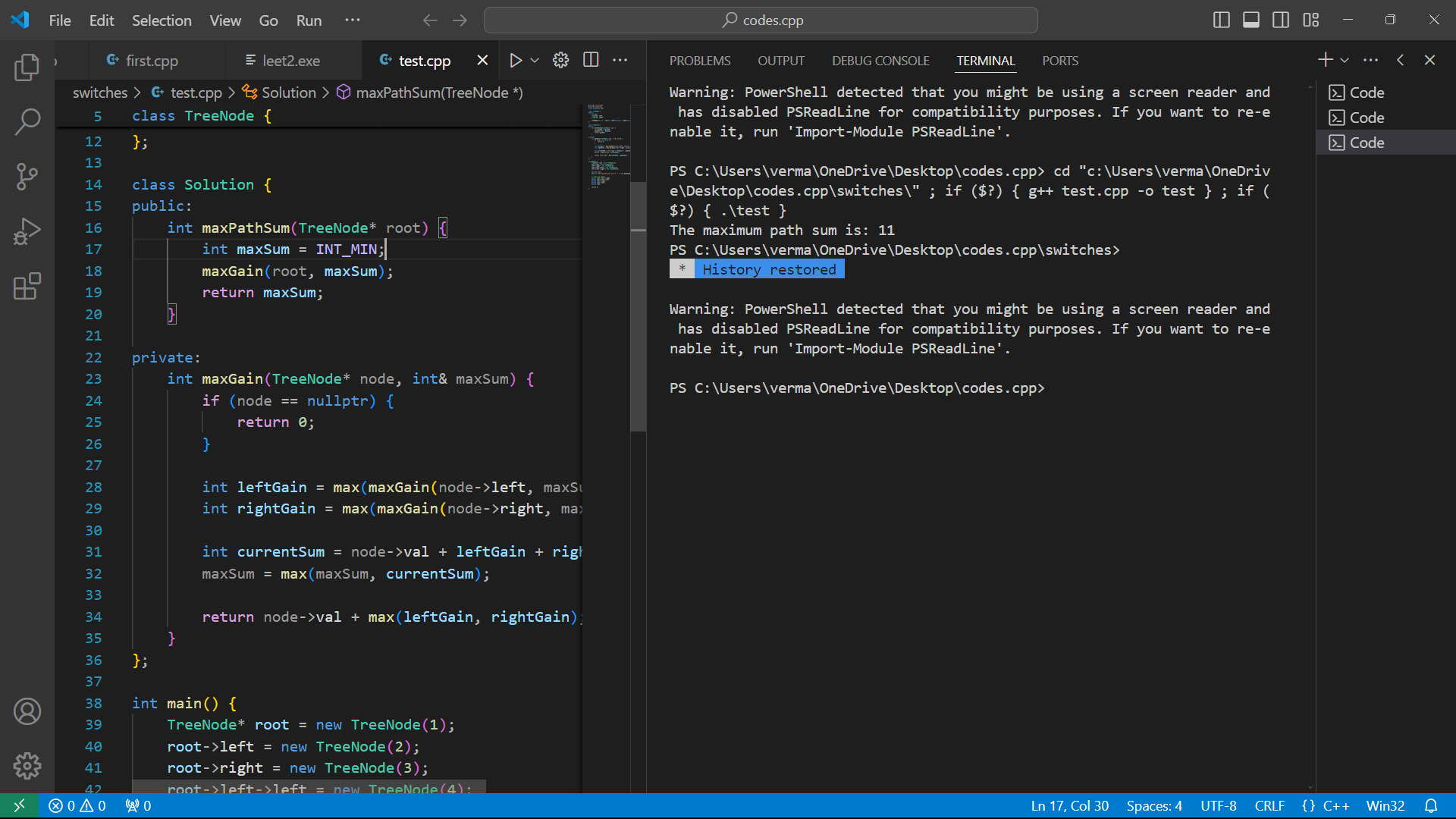
delete root->left->left;

delete root->right;

delete root->left;

delete root;

return 0;

} 

**SOLUTION 4**

#include <iostream>

#include <sstream>

#include <string>

#include <vector>

#include <queue>

using namespace std;

class TreeNode {

public:

int val;

TreeNode \*left;

TreeNode \*right;

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

};

class Codec {

public:

string serialize(TreeNode\* root) {

ostringstream out;

serialize(root, out);

return out.str();

}

TreeNode\* deserialize(const string& data) {

istringstream in(data);

return deserialize(in);

}

private:

void serialize(TreeNode\* root, ostringstream& out) {

if (!root) {

out << "# ";

return;

}

out << root->val << " ";

serialize(root->left, out);

serialize(root->right, out);

}

TreeNode\* deserialize(istringstream& in) {

string val;

in >> val;

if (val == "#") {

return NULL;

}

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

root->left = deserialize(in);

root->right = deserialize(in);

return root;

}

};

void printTree(TreeNode\* root) {

if (!root) return;

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

if (node) {

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

q.push(node->left);

q.push(node->right);

} else {

cout << "# ";

}

}

cout << endl;

}

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);

string serialized = ser.serialize(root);

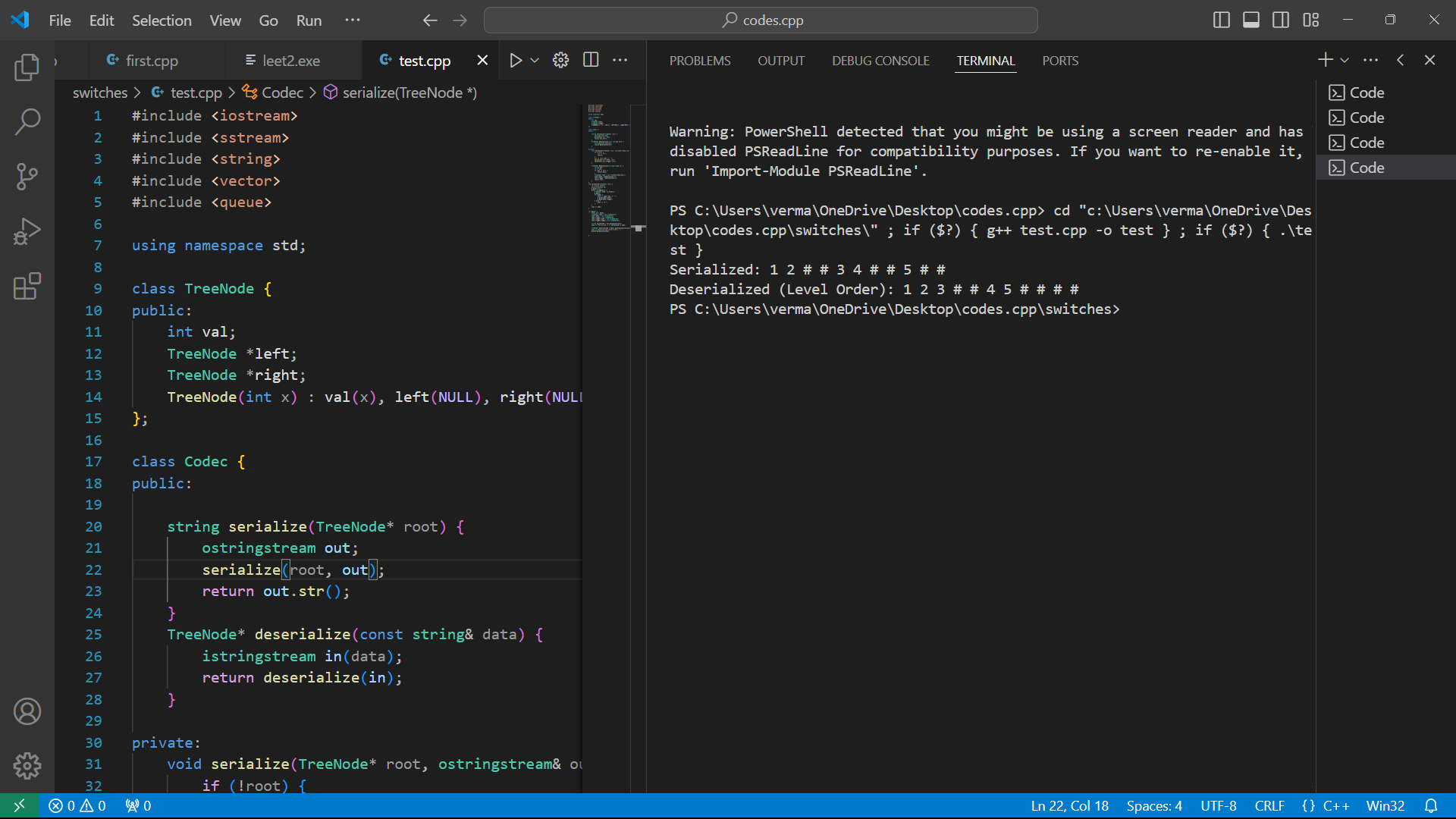
cout << "Serialized: " << serialized << endl;

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

cout << "Deserialized (Level Order): ";

printTree(deserialized);

}



**SOLUTION 5**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

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

int n = num.size();

int k=3; //amount by whuch we want to rotate the array;

k = k%n;// this is due to if the no. k exceed the size of array

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

cout<<num[i]<<" ";

}

cout<<endl;

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

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

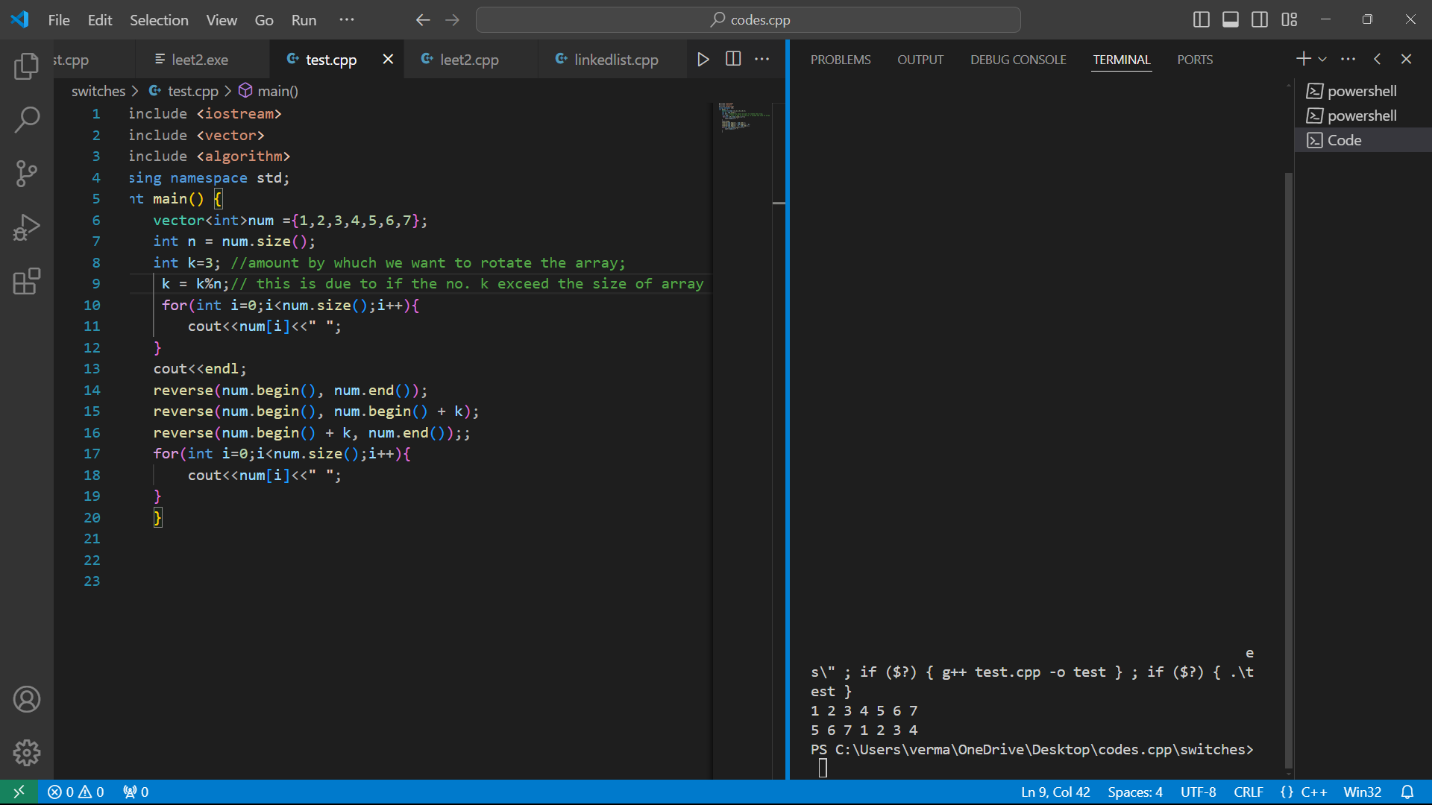
reverse(num.begin() + k, num.end());;

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

cout<<num[i]<<" ";

}

}



**SOLUTION 6**

#include<iostream>

using namespace std;

int factorial(int n){

if(n==1){

return 1;

}

return n\*factorial(n-1);

}

int main() {

int number;

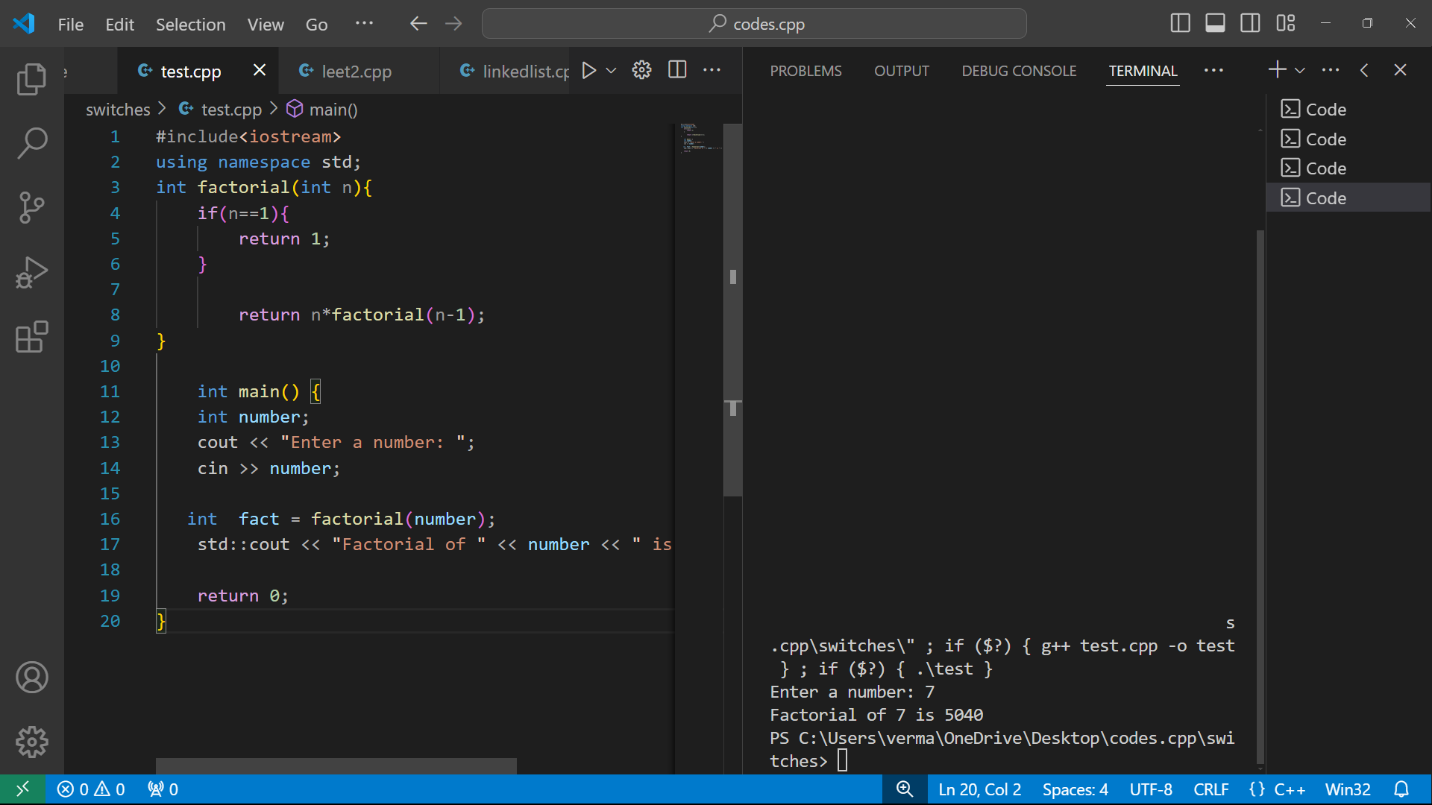
cout << "Enter a number: ";

cin >> number;

int fact = factorial(number);

std::cout << "Factorial of " << number << " is " << fact << endl;

}



**SOLUTION 7**

#include <iostream>

using namespace std;

int solve(int n) {

n = abs(n);

int sum = 0;

while (n > 0) {

sum += n % 10;

n /= 10;

}

return sum;

}

int main() {

int number;

cout << "Enter a number: ";

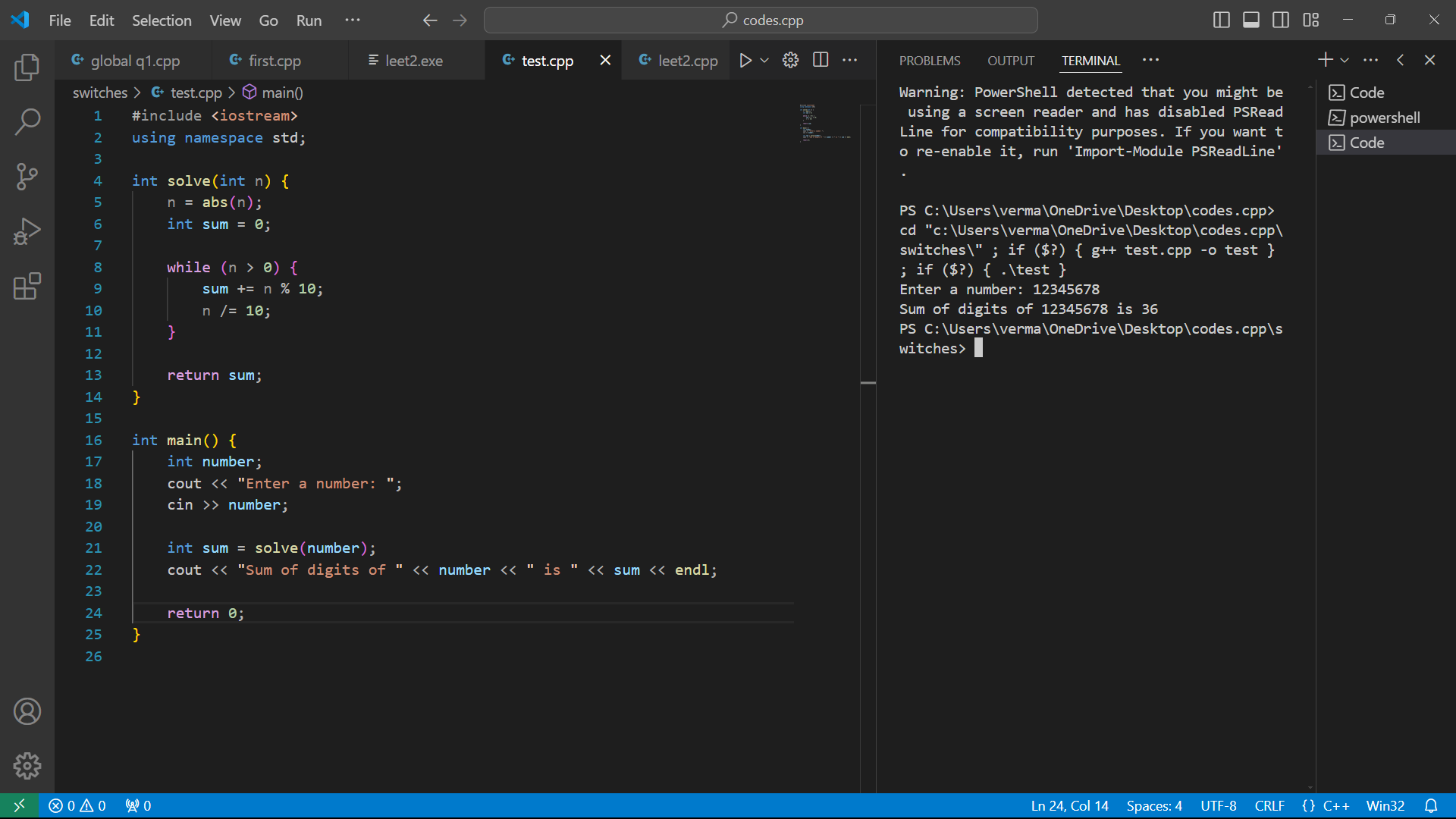
cin >> number;

int sum = solve(number);

cout << "Sum of digits of " << number << " is " << sum << endl;

return 0;

}



**SOLUTION 8**

#include<iostream>

using namespace std;

int gcd(int a, int b) {

if(a==0)

return b;

if(b==0)

return a;

while(a != b) {

if(a>b)

{

a = a-b;

}

else{

b = b-a;

}

}

return a;

}

int main() {

int a,b;

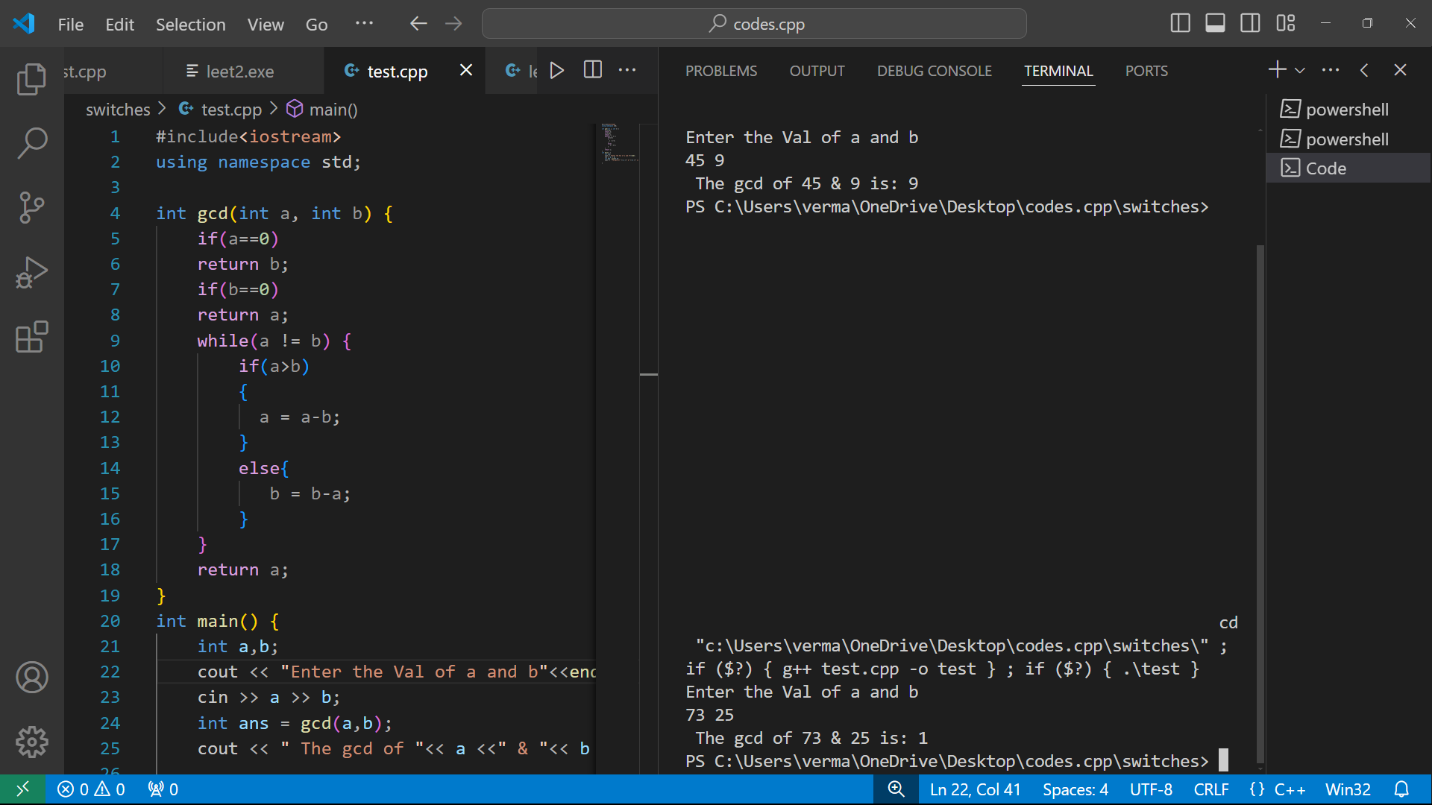
cout << "Enter the Val of a and b"<<endl;

cin >> a >> b;

int ans = gcd(a,b);

cout << " The gcd of "<< a <<" & "<< b <<" is: "<<ans<<endl;

}



**SOLUTION 9**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int maxdif(const vector<int>& arr) {

if (arr.size() < 2) {

cerr << " array must contain at least two elements." << endl;

return -1;

}

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

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

return maxElement - minElement;

}

int main() {

vector<int> arr;

int n, element;

cout << "Enter the size of array ";

cin >> n;

cout << "Enter the elements";

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

cin >> element;

arr.push\_back(element);

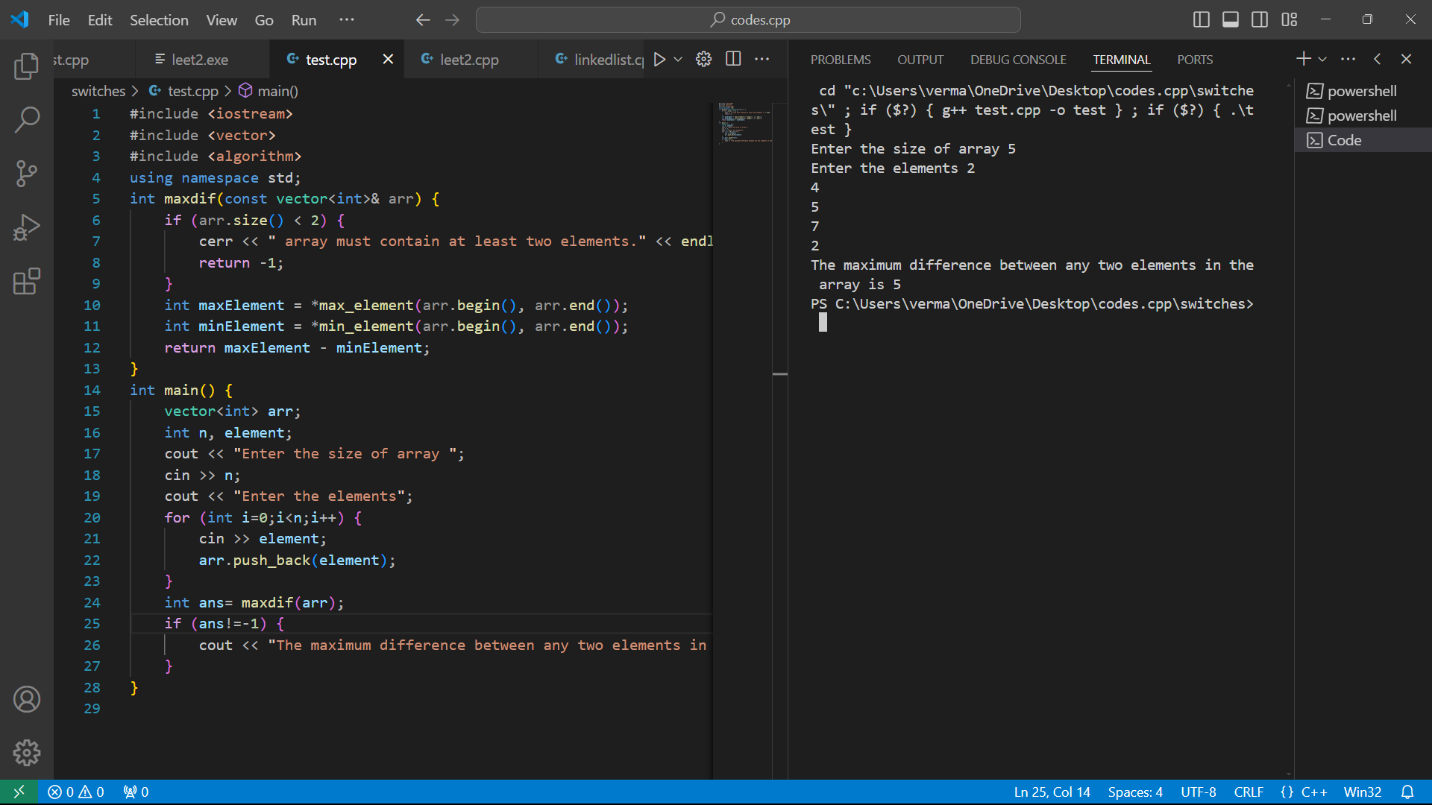
}

int ans= maxdif(arr);

if (ans!=-1) {

cout << "The maximum difference between any two elements in the array is "<<ans<<endl;

} }



**SOLUTION 10**

#include <iostream>

#include <string>

using namespace std;

bool havealphabets(const string& str) {

for (char c : str) {

if (!((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))) {

return false;

}

}

return true;

}

int main() {

string s;

cout << "Enter a string: ";

cin >> s;

if (havealphabets(s)) {

cout<<"string have alphabetic characters"<<endl;

}

else {

cout<<"string do not have alphabetic characters"<<endl;

}

}

