

Name: Tamoor Akram

Roll No: SU92-BSSEM-S24-012

Subject: DSA LAB

Submitted to: Rasikh Ali

**Task 10**

#include <iostream>

using namespace std;

class StackArray

{

int arr[100];

int top;

public:

StackArray() { top = -1; }

void push(int val) {

if (top < 99) arr[++top] = val;

else cout << "Stack Overflow\n";

}

void pop()

{

if (top >= 0) top--;

else cout << "Stack Underflow\n";

}

void display()

{

if (top == -1) {

cout << "Stack is Empty\n";

return;

}

for (int i = top; i >= 0; i--)

cout << arr[i] << " ";

cout << "\n";

}

};

struct Node

{

int data;

Node\* next;

Node(int val) {

data = val;

next = nullptr;

}

};

class StackLinkedList

{

Node\* top;

public:

StackLinkedList() { top = nullptr; }

void push(int val) {

Node\* newNode = new Node(val);

newNode->next = top;

top = newNode;

}

void pop()

{

if (top) {

Node\* temp = top;

top = top->next;

delete temp;

}

else cout << "Stack Underflow\n";

}

void display()

{

if (!top) {

cout << "Stack is Empty\n";

return;

}

Node\* temp = top;

while (temp) {

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

temp = temp->next;

}

cout << "\n";

}

};

int main()

{

StackArray sa;

sa.push(10);

sa.push(20);

sa.push(30);

sa.display();

sa.pop();

sa.display();

StackLinkedList sl;

sl.push(100);

sl.push(200);

sl.push(300);

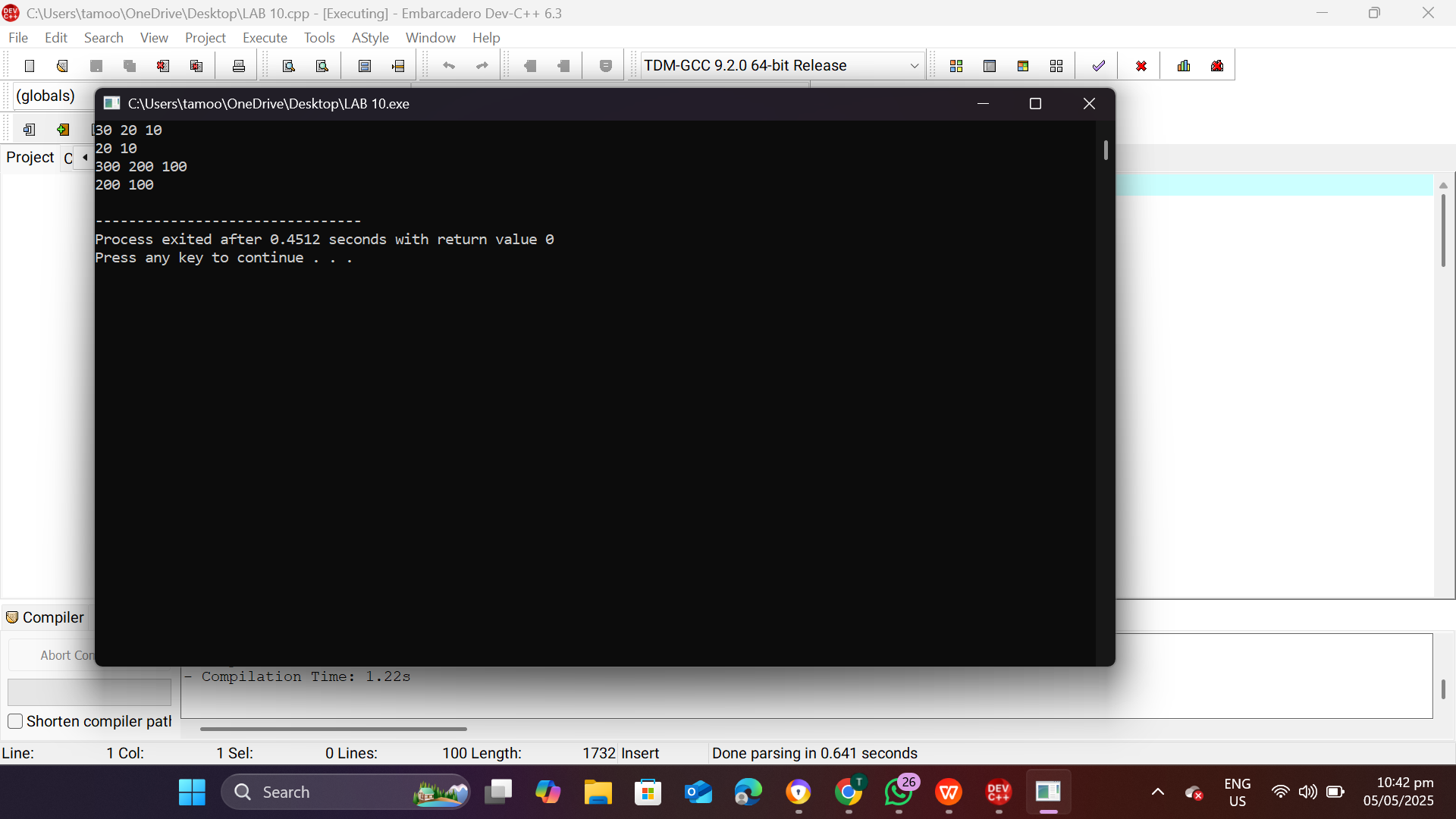
sl.display();

sl.pop();

sl.display();

return 0;

}



**Task 11**

#include <iostream>

using namespace std;

class QueueArray

{

int arr[100];

int front, rear;

public:

QueueArray() {

front = -1;

rear = -1;

}

void enqueue(int val) {

if (rear == 99) return;

if (front == -1) front = 0;

arr[++rear] = val;

}

void dequeue()

{

if (front == -1 || front > rear) {

front = rear = -1;

return;

}

front++;

}

void display()

{

if (front == -1 || front > rear) {

cout << "Array Queue is Empty\n";

return;

}

for (int i = front; i <= rear; i++)

cout << arr[i] << " ";

cout << "\n";

}

};

struct Node

{

int data;

Node\* next;

Node(int val) {

data = val;

next = nullptr;

}

};

class QueueLinkedList

{

Node\* front;

Node\* rear;

public:

QueueLinkedList() {

front = rear = nullptr;

}

void enqueue(int val)

{

Node\* newNode = new Node(val);

if (!rear) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

void dequeue() {

if (!front) return;

Node\* temp = front;

front = front->next;

if (!front) rear = nullptr;

delete temp;

}

void display()

{

if (!front) {

cout << "Linked List Queue is Empty\n";

return;

}

Node\* temp = front;

while (temp) {

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

temp = temp->next;

}

cout << "\n";

}

};

int main()

{

QueueArray qa;

qa.enqueue(1);

qa.enqueue(2);

qa.enqueue(3);

qa.display();

qa.dequeue();

qa.display();

QueueLinkedList ql;

ql.enqueue(10);

ql.enqueue(20);

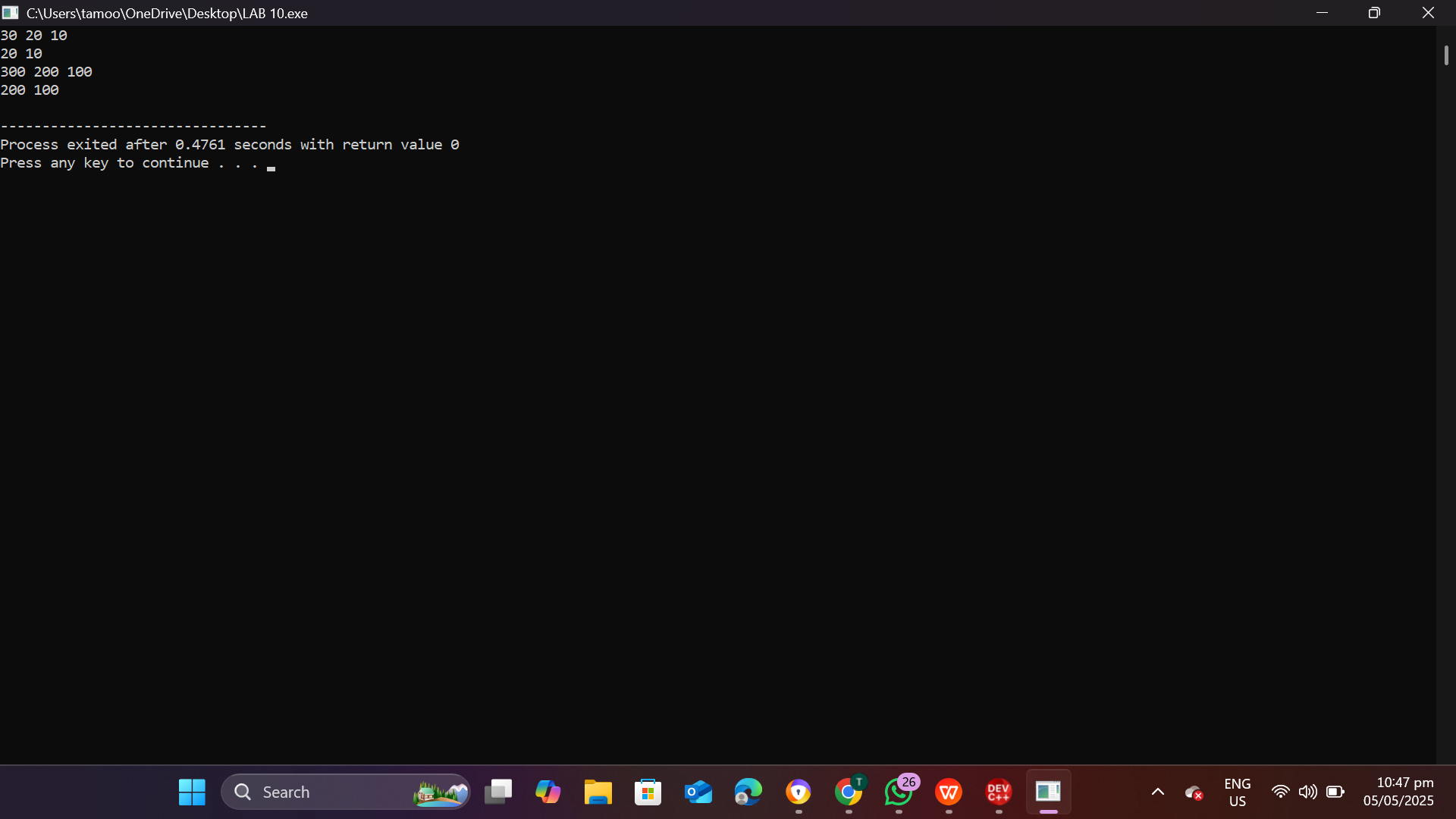
ql.enqueue(30);

ql.display();

ql.dequeue();

ql.display();

return 0;

}

**Task 12**

#include <iostream>

using namespace std;

struct BSTNode

{

int data;

BSTNode\* left;

BSTNode\* right;

BSTNode(int val) {

data = val;

left = right = nullptr;

}

};

class BST

{

public:

BSTNode\* insert(BSTNode\* root, int val) {

if (!root) return new BSTNode(val);

if (val < root->data)

root->left = insert(root->left, val);

else

root->right = insert(root->right, val);

return root;

}

void inorder(BSTNode\* root) {

if (!root) return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

};

struct AVLNode

{

int data;

AVLNode\* left;

AVLNode\* right;

int height;

AVLNode(int val) {

data = val;

left = right = nullptr;

height = 1;

}

};

class AVL

{

public:

int height(AVLNode\* n) {

return n ? n->height : 0;

}

int getBalance(AVLNode\* n) {

return n ? height(n->left) - height(n->right) : 0;

}

AVLNode\* rightRotate(AVLNode\* y)

{

AVLNode\* x = y->left;

AVLNode\* T2 = x->right;

x->right = y;

y->left = T2;

y->height = 1 + max(height(y->left), height(y->right));

x->height = 1 + max(height(x->left), height(x->right));

return x;

}

AVLNode\* leftRotate(AVLNode\* x)

{

AVLNode\* y = x->right;

AVLNode\* T2 = y->left;

y->left = x;

x->right = T2;

x->height = 1 + max(height(x->left), height(x->right));

y->height = 1 + max(height(y->left), height(y->right));

return y;

}

AVLNode\* insert(AVLNode\* root, int val)

{

if (!root) return new AVLNode(val);

if (val < root->data)

root->left = insert(root->left, val);

else if (val > root->data)

root->right = insert(root->right, val);

else

return root;

root->height = 1 + max(height(root->left), height(root->right));

int balance = getBalance(root);

if (balance > 1 && val < root->left->data)

return rightRotate(root);

if (balance < -1 && val > root->right->data)

return leftRotate(root);

if (balance > 1 && val > root->left->data) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && val < root->right->data) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

void inorder(AVLNode\* root) {

if (!root) return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

};

int main()

{

BST bst;

BSTNode\* bstRoot = nullptr;

bstRoot = bst.insert(bstRoot, 30);

bst.insert(bstRoot, 20);

bst.insert(bstRoot, 40);

bst.insert(bstRoot, 10);

cout << "BST In-order: ";

bst.inorder(bstRoot);

cout << "\n";

AVL avl;

AVLNode\* avlRoot = nullptr;

avlRoot = avl.insert(avlRoot, 30);

avlRoot = avl.insert(avlRoot, 20);

avlRoot = avl.insert(avlRoot, 40);

avlRoot = avl.insert(avlRoot, 10);

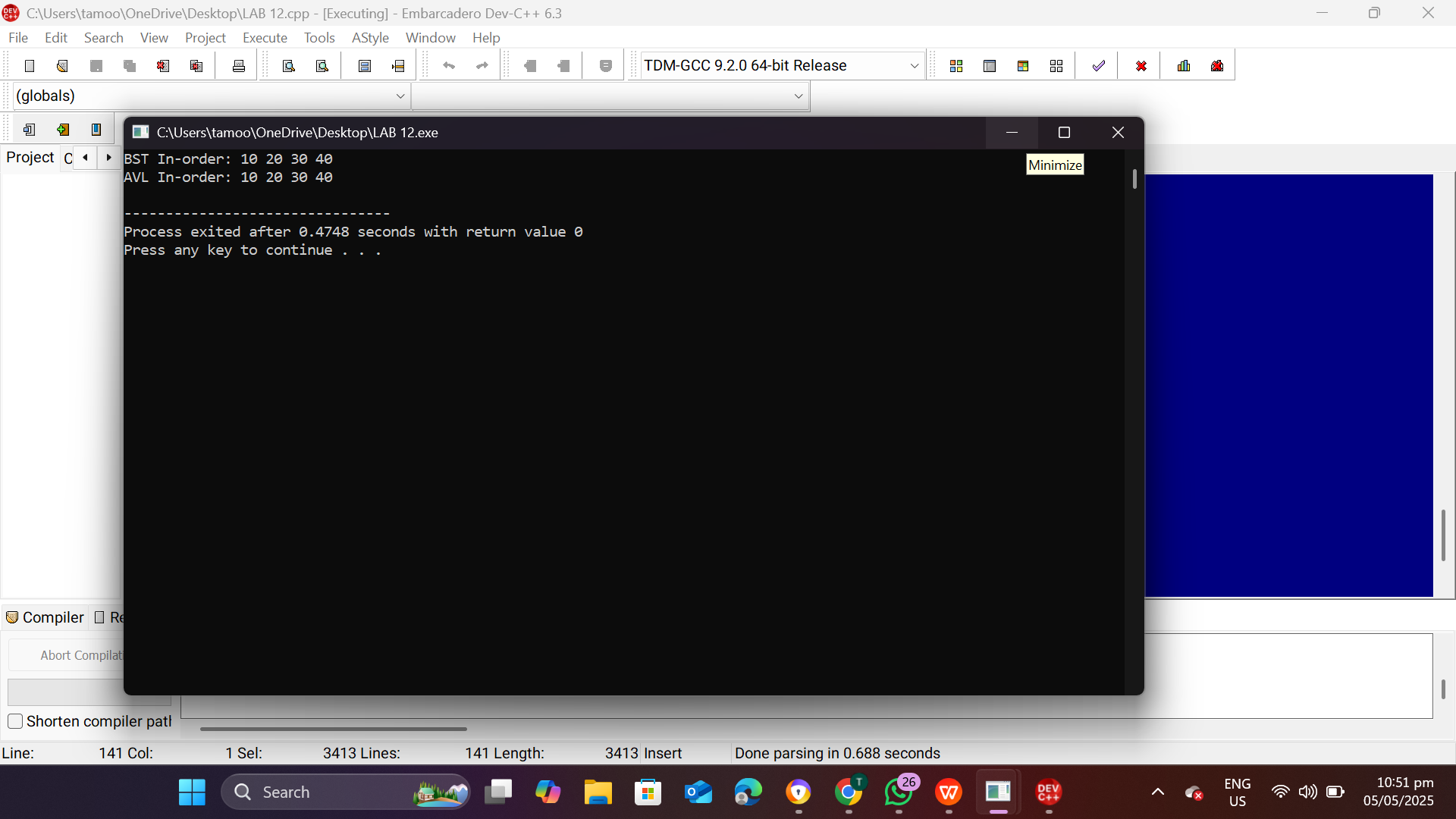
cout << "AVL In-order: ";

avl.inorder(avlRoot);

cout << "\n";

return 0;

}



**Task 13**

#include <iostream>

#include <queue>

#include <list>

#include <unordered\_map>

#include <unordered\_set>

using namespace std;

class Node

{

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void dfsTree(Node\* root)

{

if (root == nullptr) return;

cout << root->data << " ";

dfsTree(root->left);

dfsTree(root->right);

}

void bfsTree(Node\* root)

{

if (root == nullptr) return;

queue<Node\*> q;

q.push(root);

while (!q.empty()) {

Node\* temp = q.front();

q.pop();

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

if (temp->left) q.push(temp->left);

if (temp->right) q.push(temp->right);

}

}

class Graph

{

public:

unordered\_map<int, list<int>> adj;

void addEdge(int u, int v) {

adj[u].push\_back(v);

adj[v].push\_back(u);

}

void dfsGraph(int start, unordered\_set<int>& visited)

{

visited.insert(start);

cout << start << " ";

for (int neighbor : adj[start]) {

if (visited.find(neighbor) == visited.end()) {

dfsGraph(neighbor, visited);

}

}

}

void bfsGraph(int start)

{

queue<int> q;

unordered\_set<int> visited;

q.push(start);

visited.insert(start);

while (!q.empty()) {

int curr = q.front();

q.pop();

cout << curr << " ";

for (int neighbor : adj[curr]) {

if (visited.find(neighbor) == visited.end()) {

visited.insert(neighbor);

q.push(neighbor);

}

}

}

}

};

int main()

{

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

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

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

cout << "DFS Tree: ";

dfsTree(root);

cout << "\n";

cout << "BFS Tree: ";

bfsTree(root);

cout << "\n";

Graph g;

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(2, 4);

g.addEdge(2, 5);

unordered\_set<int> visitedNodes;

cout << "DFS Graph: ";

g.dfsGraph(1, visitedNodes);

cout << "\n";

cout << "BFS Graph: ";

g.bfsGraph(1);

cout << "\n";

delete root->left->right;

delete root->left->left;

delete root->left;

delete root->right;

delete root;

return 0;

}

