**1. Finding maximum from array.**

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter the size of the array: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements: ";

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

cin >> arr[i];

}

int maxVal = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > maxVal) {

maxVal = arr[i];

}

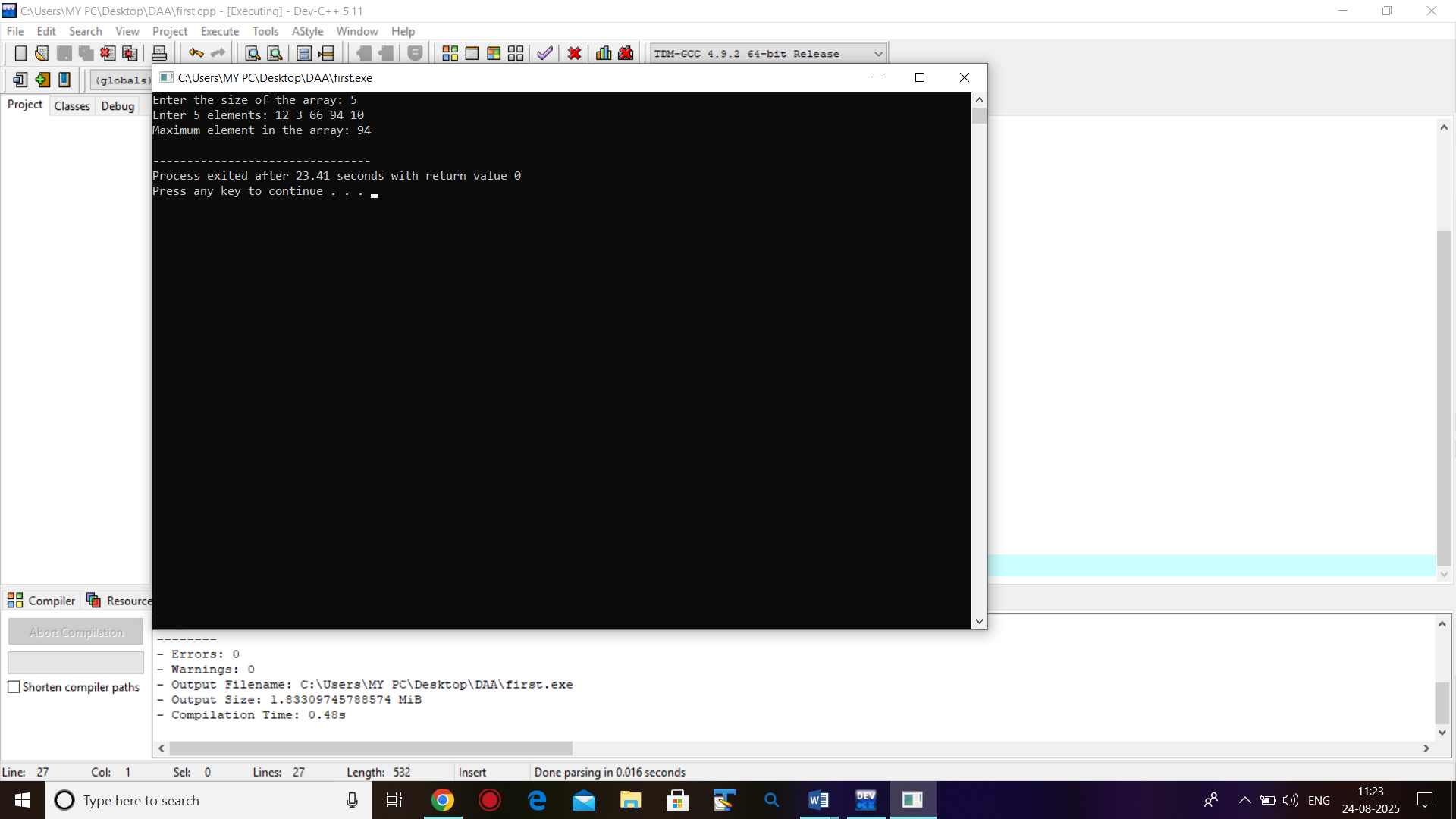
}

cout << "Maximum element in the array: " << maxVal << endl;

return 0;

}

**Output:**



**2. Binomial Coefficient B(n, m)= B(n-1, m-1)+B(n-1,m), B(n ,n)=B(n,0)=1**

#include <iostream>

using namespace std;

int binomialCoeff(int n, int m) {

if (m == 0 || m == n)

return 1;

return binomialCoeff(n - 1, m - 1) + binomialCoeff(n - 1, m);

}

int main() {

int n, m;

cout << "Enter n and m: ";

cin >> n >> m;

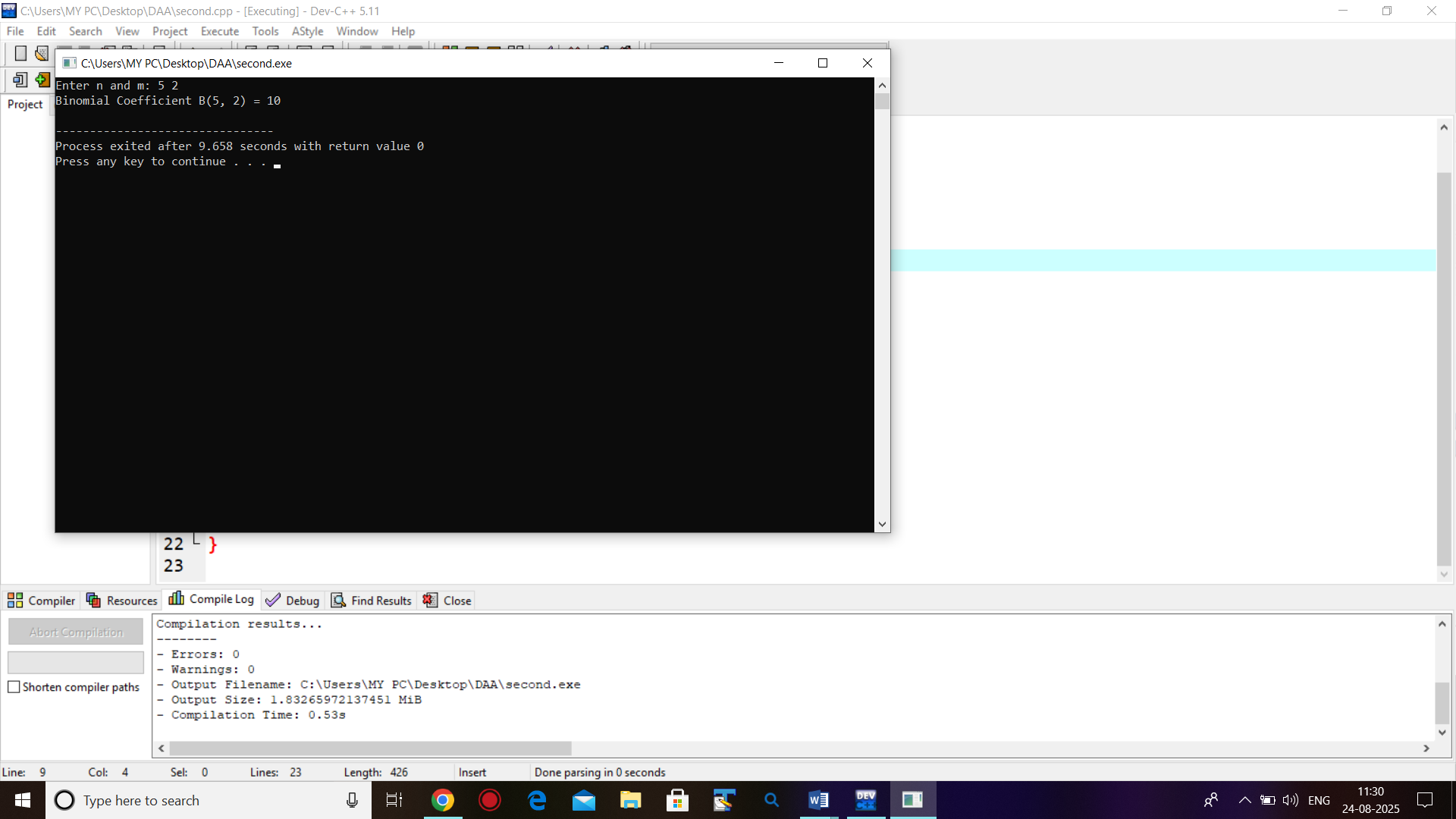
cout << "Binomial Coefficient B(" << n << ", " << m << ") = "

<< binomialCoeff(n, m) << endl;

return 0;

}

**Output:**



**3. Searching element from array.**

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter size of array: ";

cin >> n;

int arr[n];

cout << "Enter elements:\n";

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

cin >> arr[i];

int key;

cout << "Enter element to search: ";

cin >> key;

int index = -1;

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

if (arr[i] == key) {

index = i;

break;

}

}

if (index != -1)

cout << "Element found at index " << index << endl;

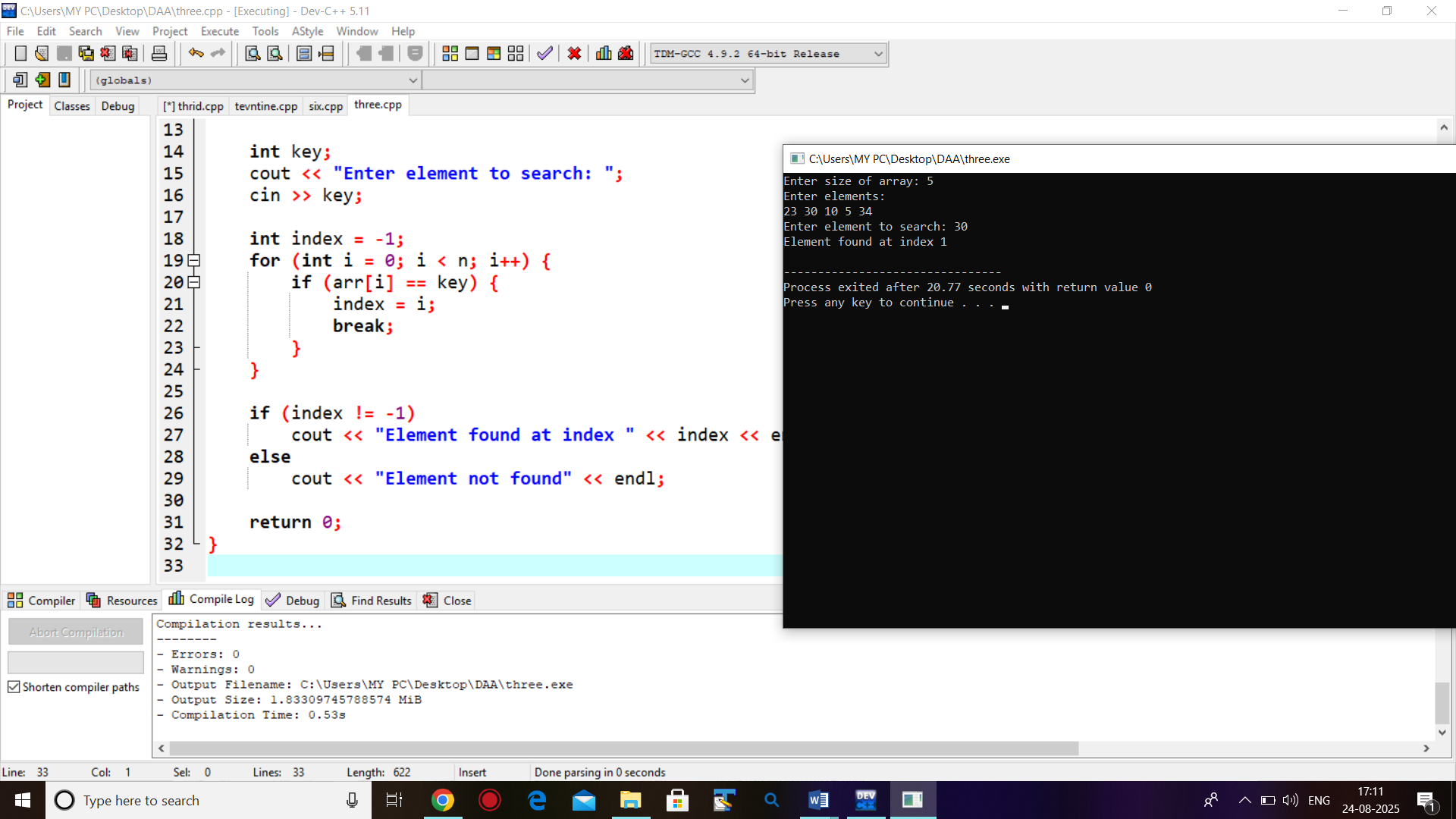
else

cout << "Element not found" << endl;

return 0;

}

**Output:**

****

**4. Write a program for creating Max/Min. heap using INSERT.**

#include <iostream>

using namespace std;

int maxHeap[100], minHeap[100];

int maxSize = 0, minSize = 0;

void insertMax(int key) {

int i = maxSize;

maxHeap[maxSize++] = key;

while (i != 0 && maxHeap[i] > maxHeap[(i - 1) / 2]) {

swap(maxHeap[i], maxHeap[(i - 1) / 2]);

i = (i - 1) / 2;

}

}

void insertMin(int key) {

int i = minSize;

minHeap[minSize++] = key;

while (i != 0 && minHeap[i] < minHeap[(i - 1) / 2]) {

swap(minHeap[i], minHeap[(i - 1) / 2]);

i = (i - 1) / 2;

}

}

void printHeap(int heap[], int size) {

for (int i = 0; i < size; i++) cout << heap[i] << " ";

cout << endl;

}

int main() {

int n, x;

cout << "Enter number of elements: ";

cin >> n;

cout << "Enter elements:\n";

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

cin >> x;

insertMax(x);

insertMin(x);

}

cout << "Max Heap: ";

printHeap(maxHeap, maxSize);

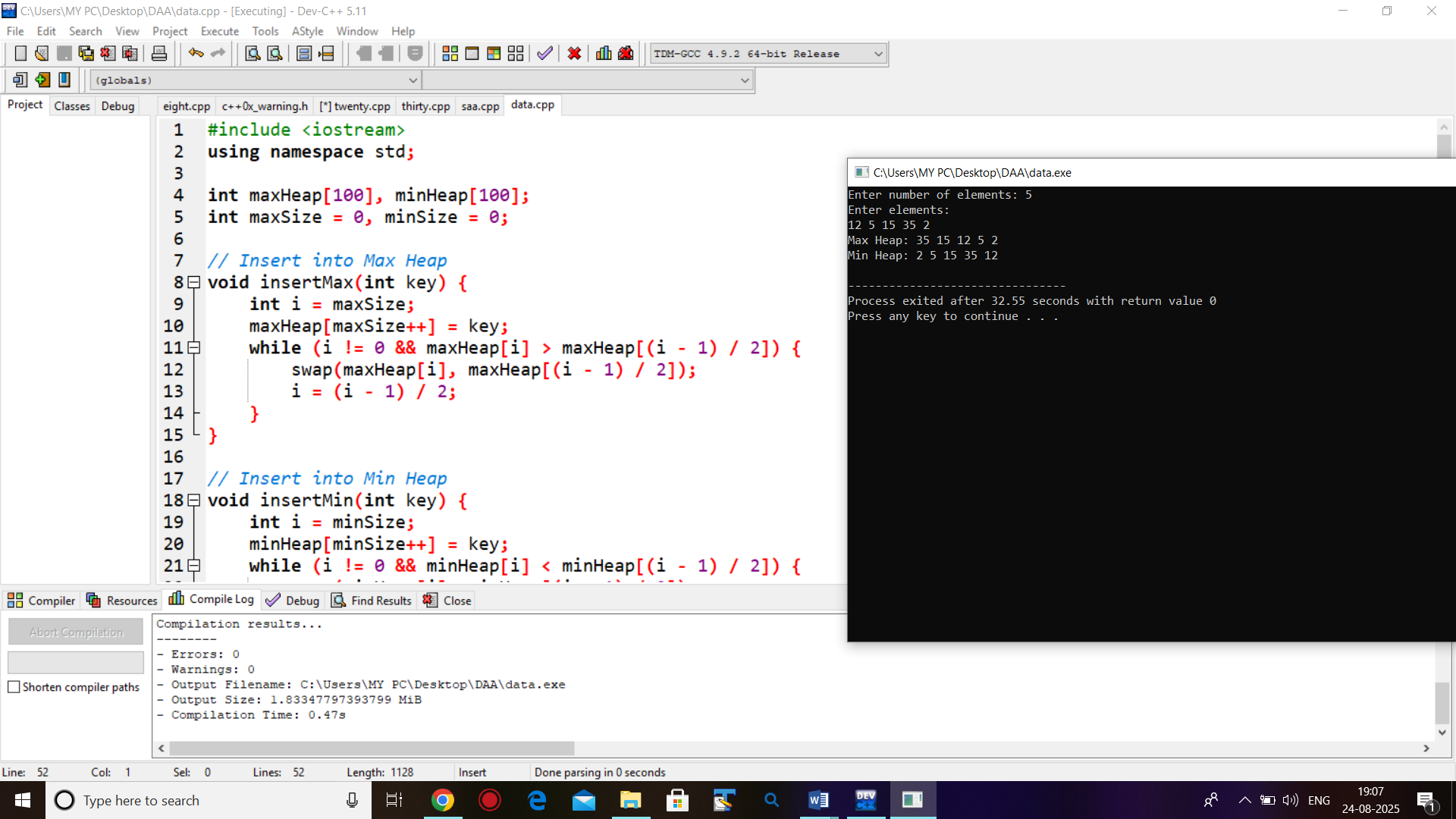
cout << "Min Heap: ";

printHeap(minHeap, minSize);

return 0;

}

**Output:**

****

**5. Write a program for creating Max/Min. heap using ADJUST/HEAPIFY.**

#include <iostream>

using namespace std;

void maxHeapify(int arr[], int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(arr[i], arr[largest]);

maxHeapify(arr, n, largest);

}

}

void minHeapify(int arr[], int n, int i) {

int smallest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] < arr[smallest])

smallest = left;

if (right < n && arr[right] < arr[smallest])

smallest = right;

if (smallest != i) {

swap(arr[i], arr[smallest]);

minHeapify(arr, n, smallest);

}

}

void buildMaxHeap(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--)

maxHeapify(arr, n, i);

}

void buildMinHeap(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--)

minHeapify(arr, n, i);

}

void printHeap(int arr[], int n) {

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

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

cout << endl;

}

int main() {

int n;

cout << "Enter number of elements: ";

cin >> n;

int arr[n];

cout << "Enter elements:\n";

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

cin >> arr[i];

int maxHeap[n], minHeap[n];

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

maxHeap[i] = arr[i];

minHeap[i] = arr[i];

}

buildMaxHeap(maxHeap, n);

buildMinHeap(minHeap, n);

cout << "Max Heap: ";

printHeap(maxHeap, n);

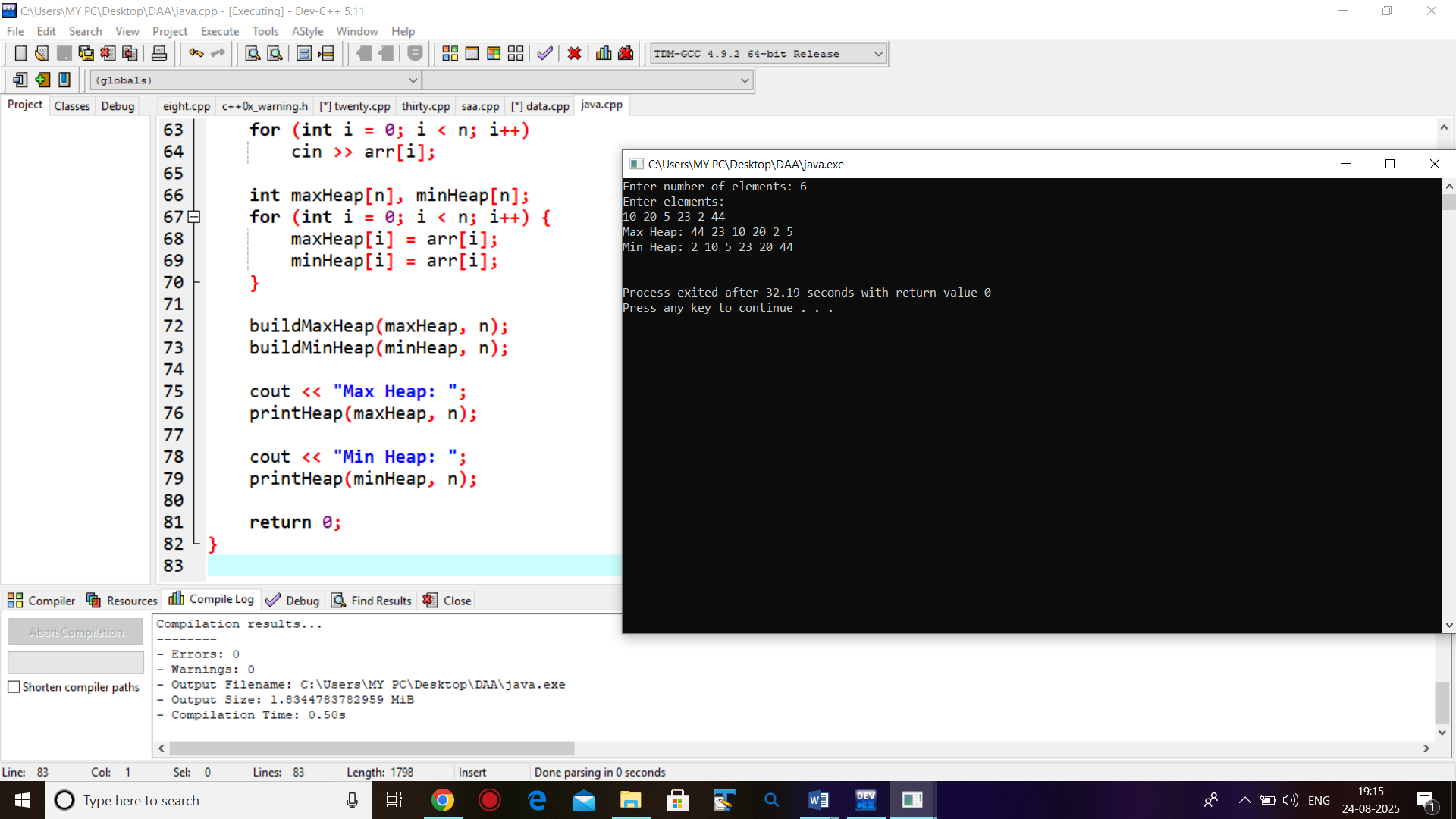
cout << "Min Heap: ";

printHeap(minHeap, n);

return 0;

}

**Output:**



**6. Write a program for sorting given array in ascending/descending order**

**with n=1000, 2000, 3000. Find exact time of execution using Heap Sort.**

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

void maxHeapify(int arr[], int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(arr[i], arr[largest]);

maxHeapify(arr, n, largest);

}

}

void minHeapify(int arr[], int n, int i) {

int smallest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] < arr[smallest])

smallest = left;

if (right < n && arr[right] < arr[smallest])

smallest = right;

if (smallest != i) {

swap(arr[i], arr[smallest]);

minHeapify(arr, n, smallest);

}

}

void heapSortAscending(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--)

maxHeapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {

swap(arr[0], arr[i]); // Move max to end

maxHeapify(arr, i, 0);

}

}

void heapSortDescending(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--)

minHeapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {

swap(arr[0], arr[i]); // Move min to end

minHeapify(arr, i, 0);

}

}

void generateArray(int arr[], int n) {

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

arr[i] = rand() % 10000;

}

int main() {

srand(time(0));

int sizes[] = {1000, 2000, 3000};

for (int s = 0; s < 3; s++) {

int n = sizes[s];

int\* arr = new int[n];

generateArray(arr, n);

int\* arrAsc = new int[n];

for (int i = 0; i < n; i++) arrAsc[i] = arr[i];

clock\_t startAsc = clock();

heapSortAscending(arrAsc, n);

clock\_t endAsc = clock();

double timeAsc = double(endAsc - startAsc) / CLOCKS\_PER\_SEC;

int\* arrDesc = new int[n];

for (int i = 0; i < n; i++) arrDesc[i] = arr[i];

clock\_t startDesc = clock();

heapSortDescending(arrDesc, n);

clock\_t endDesc = clock();

double timeDesc = double(endDesc - startDesc) / CLOCKS\_PER\_SEC;

cout << "Array Size: " << n << endl;

cout << "Heap Sort (Ascending) Time: " << timeAsc << " seconds" << endl;

cout << "Heap Sort (Descending) Time: " << timeDesc << " seconds" << endl;

cout << "--------------------------------------------" << endl;

delete[] arr;

delete[] arrAsc;

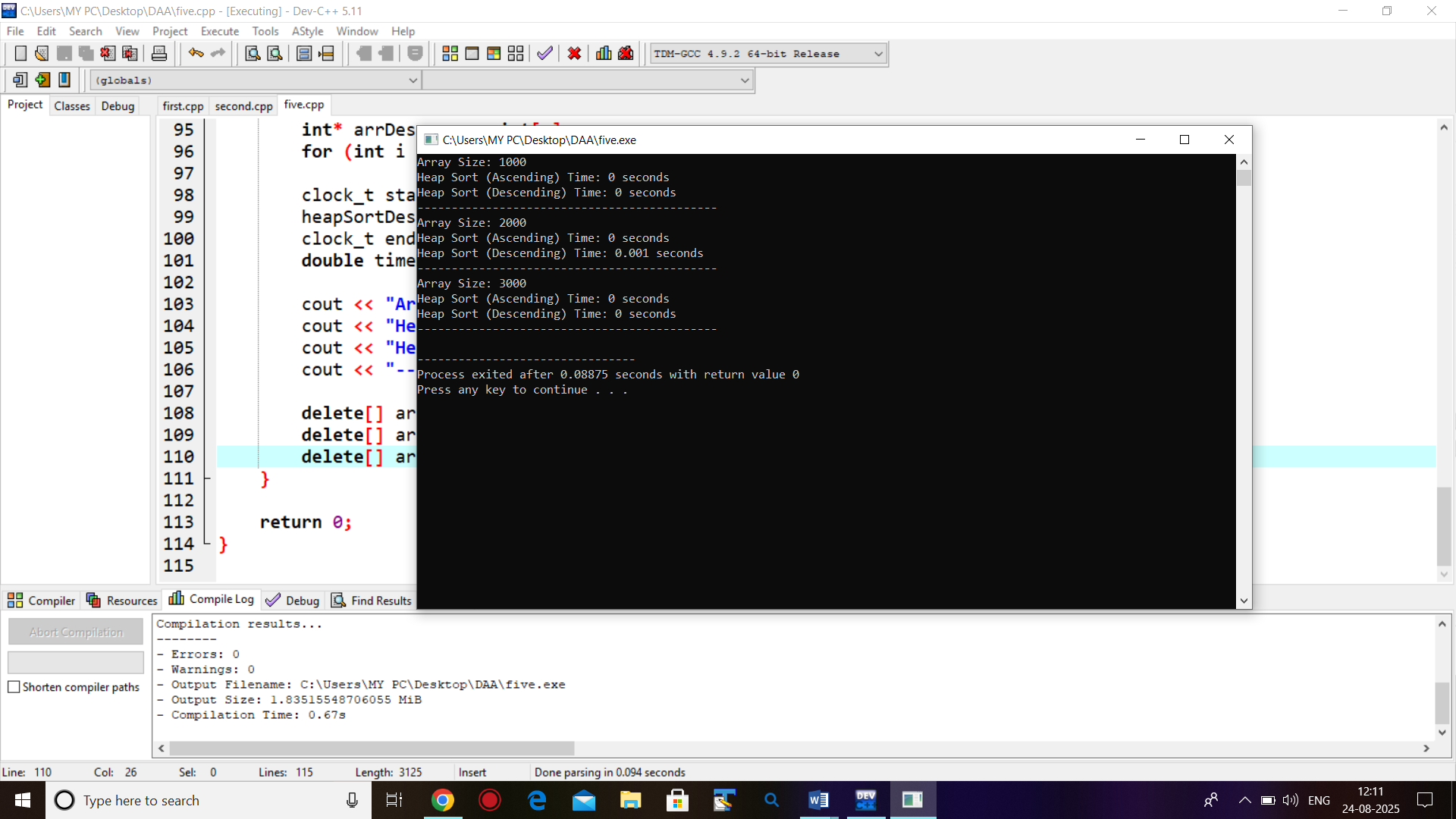
delete[] arrDesc;

}

return 0;

}

**Output:**



**7. Write a program to implement Weighted UNION and Collapsing FIND operations.**

#include <iostream>

using namespace std;

const int MAX = 100;

int parent[MAX], size[MAX];

void makeSet(int n) {

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

parent[i] = i;

size[i] = 1;

}

}

int find(int x) {

if (parent[x] != x)

parent[x] = find(parent[x]);

return parent[x];

}

void unionSets(int x, int y) {

int rootX = find(x);

int rootY = find(y);

if (rootX == rootY) return;

if (size[rootX] < size[rootY]) {

parent[rootX] = rootY;

size[rootY] += size[rootX];

} else {

parent[rootY] = rootX;

size[rootX] += size[rootY];

}

}

int main() {

int n;

cout << "Enter number of elements: ";

cin >> n;

makeSet(n);

unionSets(0, 1);

unionSets(2, 3);

unionSets(1, 2);

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

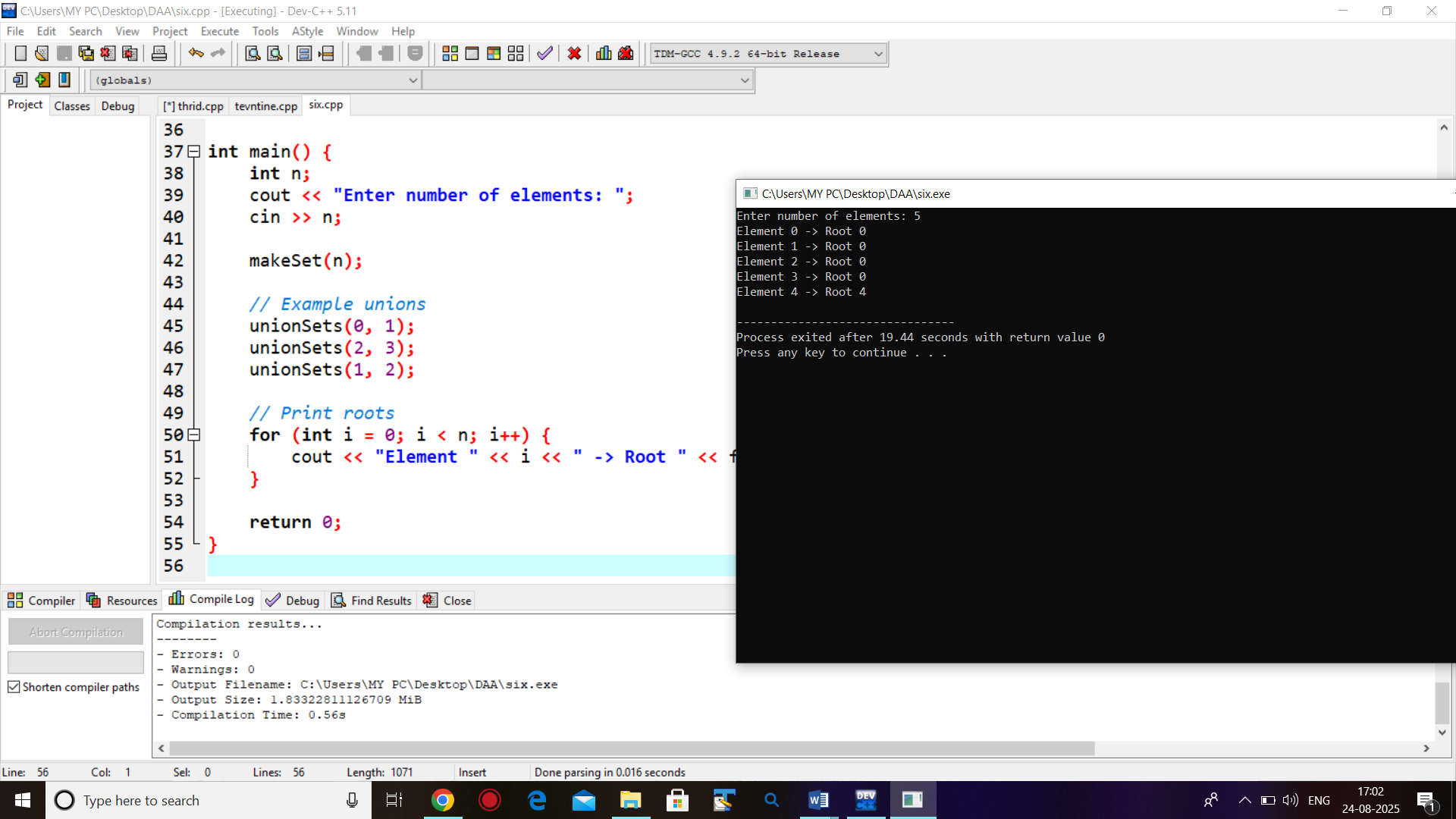
cout << "Element " << i << " -> Root " << find(i) << endl;

}

return 0;

}

**Output:**



**8. Write a program for searching element form given array using search form =1000, 2000, 3000. Find exact time of execution.**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main() {

srand(time(0));

int sizes[] = {1000, 2000, 3000};

int key;

cout << "Enter element to search: ";

cin >> key;

for (int s = 0; s < 3; s++) {

int n = sizes[s];

int\* arr = new int[n];

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

arr[i] = rand() % 10000;

arr[rand() % n] = key;

clock\_t start = clock();

int index = -1;

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

if (arr[i] == key) {

index = i;

break;

}

}

clock\_t end = clock();

double duration = double(end - start) / CLOCKS\_PER\_SEC \* 1000000; // microseconds

cout << "Array size " << n << ": ";

if (index != -1)

cout << "Found at index " << index;

else

cout << "Not found";

cout << ", Time = " << duration << " microseconds\n";

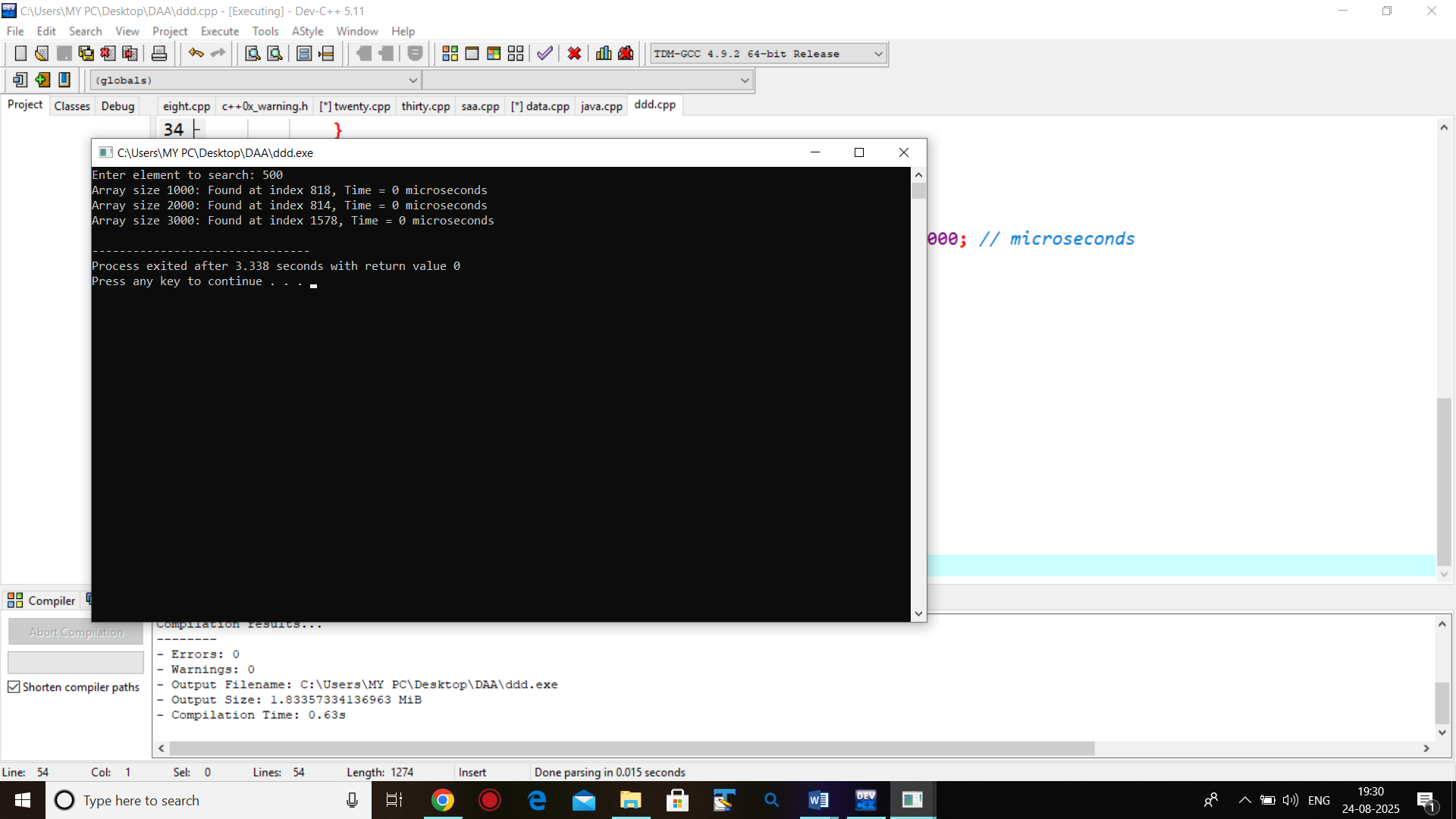
delete[] arr;

}

return 0;

}

**Output:**

****

**9. Write a program to find minimum and maximum from a given array Using MAXMIN.**

#include <iostream>

using namespace std;

struct Pair {

int min;

int max;

};

Pair MAXMIN(int arr[], int low, int high) {

Pair result, left, right;

if (low == high) {

result.min = result.max = arr[low];

return result;

}

if (high == low + 1) {

if (arr[low] < arr[high]) {

result.min = arr[low];

result.max = arr[high];

} else {

result.min = arr[high];

result.max = arr[low];

}

return result;

}

int mid = (low + high) / 2;

left = MAXMIN(arr, low, mid);

right = MAXMIN(arr, mid + 1, high);

result.min = (left.min < right.min) ? left.min : right.min;

result.max = (left.max > right.max) ? left.max : right.max;

return result;

}

int main() {

int n;

cout << "Enter number of elements: ";

cin >> n;

int arr[n];

cout << "Enter elements: ";

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

cin >> arr[i];

}

Pair ans = MAXMIN(arr, 0, n - 1);

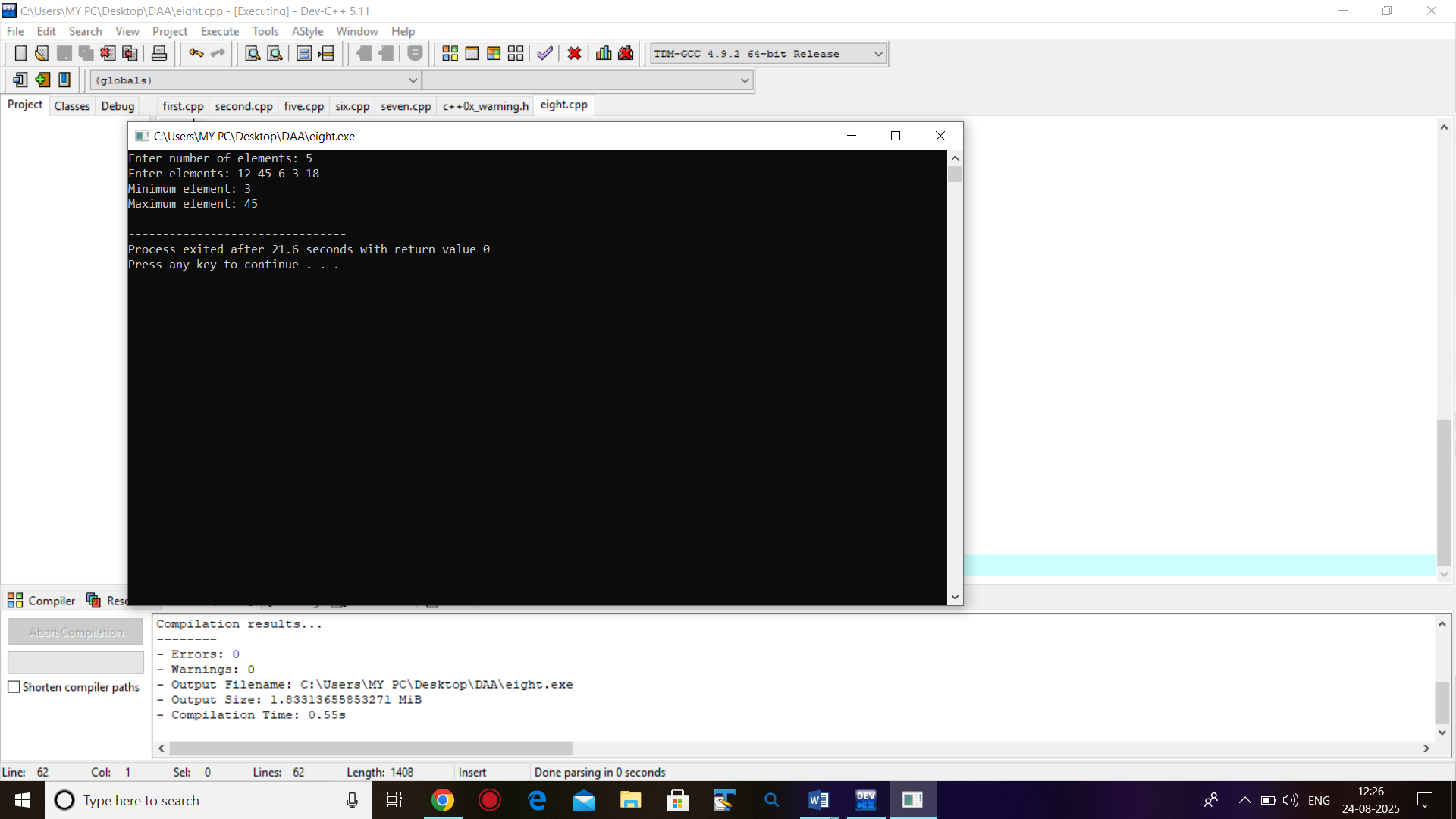
cout << "Minimum element: " << ans.min << endl;

cout << "Maximum element: " << ans.max << endl;

return 0;

}

**Output:**



**10. Write a program for sorting given array in ascending/descending order**

**with n=1000,2000,3000 find exact time of execution using – d).**

**11. Merge Sort .**

#include <iostream>

using namespace std;

void merge(int a[], int l, int m, int r) {

int i=l, j=m+1, k=0;

int temp[100];

while(i<=m && j<=r) {

if(a[i] < a[j])

temp[k++] = a[i++];

else temp[k++] = a[j++];

}

while(i<=m) temp[k++] = a[i++];

while(j<=r) temp[k++] = a[j++];

for(int p=0; p<k; p++) a[l+p] = temp[p];

}

void mergeSort(int a[], int l, int r) {

if(l<r) {

int m=(l+r)/2;

mergeSort(a,l,m);

mergeSort(a,m+1,r);

merge(a,l,m,r);

}

}

int main() {

int a[] = {5, 2, 4, 7, 1, 3, 2, 6};

int n = sizeof(a)/sizeof(a[0]);

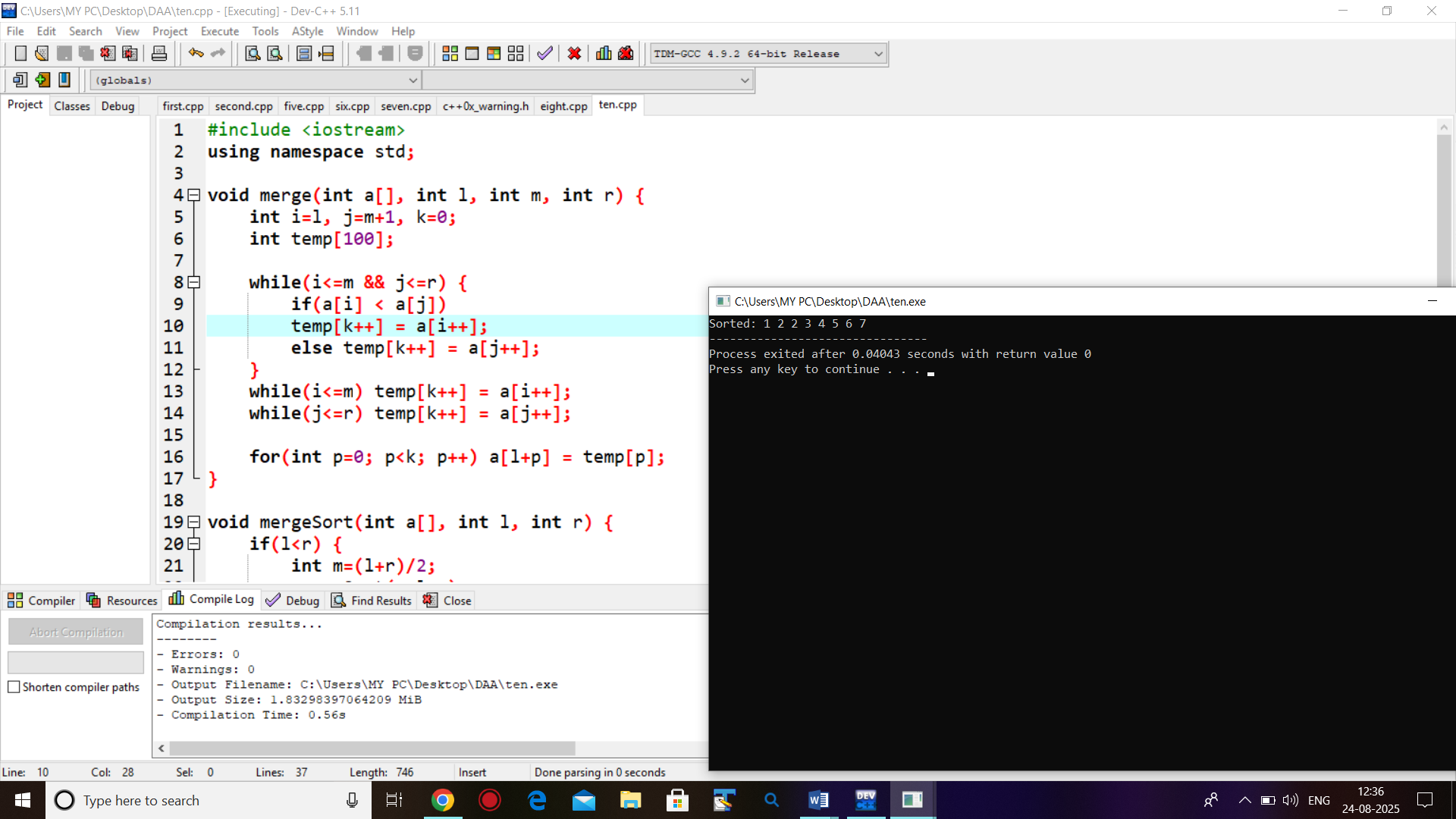
mergeSort(a,0,n-1);

cout<<"Sorted: ";

for(int i=0;i<n;i++) cout<<a[i]<<" ";

}

**Output:**



**12. Quick Sort.**

#include <iostream>

using namespace std;

int partition(int a[], int l, int h) {

int pivot = a[h]; // pivot = last element

int i = l - 1;

for (int j = l; j < h; j++) {

if (a[j] < pivot) {

i++;

swap(a[i], a[j]);

}

}

swap(a[i + 1], a[h]);

return i + 1;

}

void quickSort(int a[], int l, int h) {

if (l < h) {

int p = partition(a, l, h);

quickSort(a, l, p - 1);

quickSort(a, p + 1, h);

}

}

int main() {

int a[] = {9, 4, 7, 3, 10, 5};

int n = sizeof(a) / sizeof(a[0]);

quickSort(a, 0, n - 1);

cout << " Quick Sorted: ";

for (int i = 0; i < n; i++) cout << a[i] << " ";

}

**Output:**



**13. Write a program for matrix multiplication using Strassen’s Matrix Multiplication.**

#include <iostream>

using namespace std;

int main() {

int A[2][2], B[2][2], C[2][2];

cout << "Enter elements of 2x2 matrix A:\n";

for(int i=0;i<2;i++)

for(int j=0;j<2;j++)

cin >> A[i][j];

cout << "Enter elements of 2x2 matrix B:\n";

for(int i=0;i<2;i++)

for(int j=0;j<2;j++)

cin >> B[i][j];

int M1 = (A[0][0] + A[1][1]) \* (B[0][0] + B[1][1]);

int M2 = (A[1][0] + A[1][1]) \* B[0][0];

int M3 = A[0][0] \* (B[0][1] - B[1][1]);

int M4 = A[1][1] \* (B[1][0] - B[0][0]);

int M5 = (A[0][0] + A[0][1]) \* B[1][1];

int M6 = (A[1][0] - A[0][0]) \* (B[0][0] + B[0][1]);

int M7 = (A[0][1] - A[1][1]) \* (B[1][0] + B[1][1]);

C[0][0] = M1 + M4 - M5 + M7;

C[0][1] = M3 + M5;

C[1][0] = M2 + M4;

C[1][1] = M1 - M2 + M3 + M6;

cout << "Resultant Matrix C:\n";

for(int i=0;i<2;i++) {

for(int j=0;j<2;j++)

cout << C[i][j] << " ";

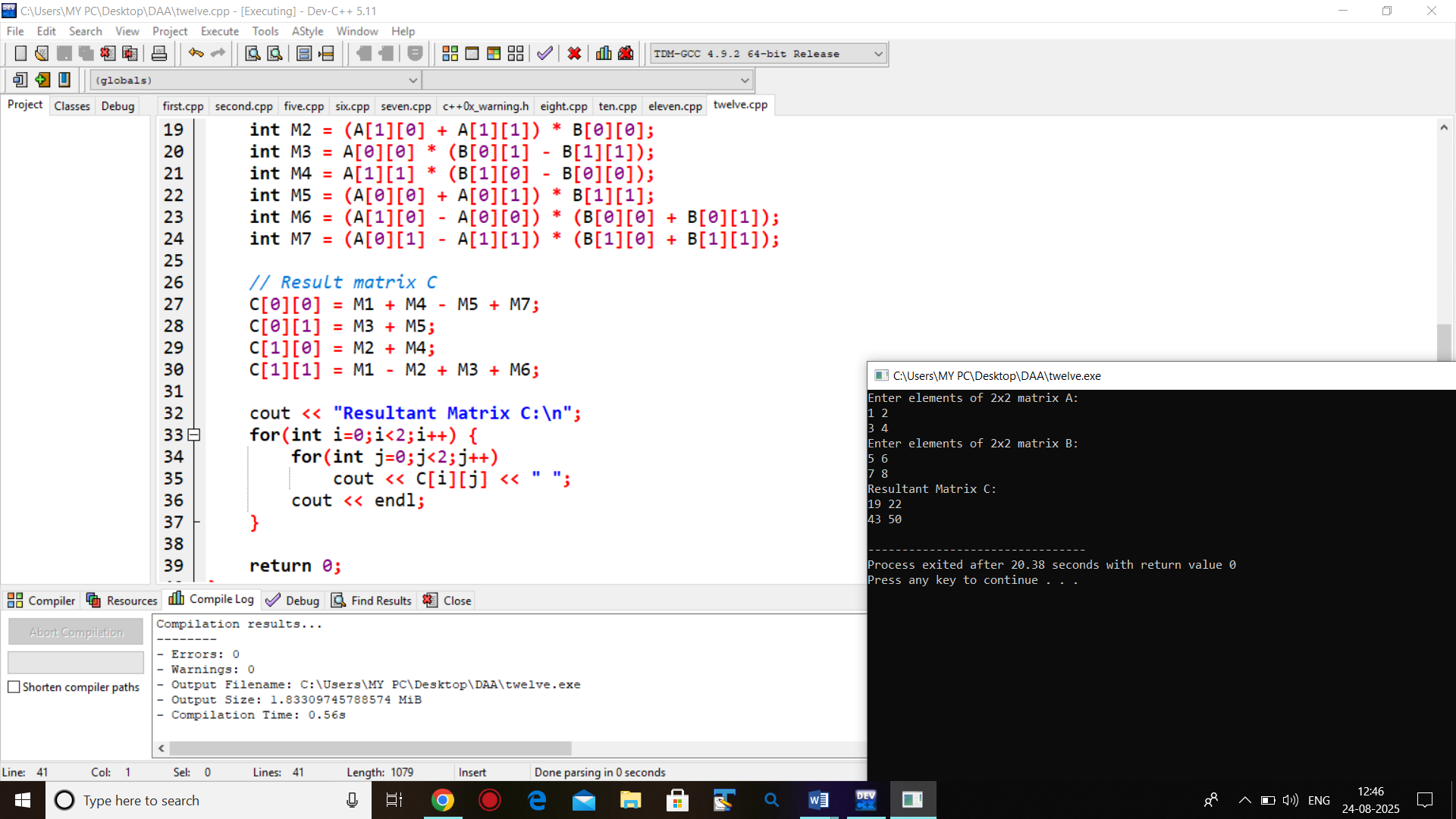
cout << endl;

}

return 0;

}

**Output:**



**14. Write a program to find solution of Fractional Knapsack instance.**

#include <iostream>

#include <algorithm>

using namespace std;

struct Item {

int value, weight;

double ratio; // value/weight

};

bool cmp(Item a, Item b) {

return a.ratio > b.ratio;

}

double fractionalKnapsack(Item items[], int n, int W) {

sort(items, items+n, cmp);

double maxValue = 0.0;

int currWeight = 0;

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

if (currWeight + items[i].weight <= W) {

currWeight += items[i].weight;

maxValue += items[i].value;

}

else {

int remain = W - currWeight;

maxValue += items[i].value \* ((double)remain / items[i].weight);

break;

}

}

return maxValue;

}

int main() {

int n, W;

cout << "Enter number of items: ";

cin >> n;

cout << "Enter knapsack capacity: ";

cin >> W;

Item items[n];

cout << "Enter value and weight of items:\n";

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

cin >> items[i].value >> items[i].weight;

items[i].ratio = (double)items[i].value / items[i].weight;

}

double result = fractionalKnapsack(items, n, W);

cout << "Maximum value in Knapsack = " << result << endl;

return 0;

}

**Output:**



**15. Write a program to find Minimum Spanning Tree using Prim’s algorithm.**

#include <iostream>

#include <climits>

using namespace std;

#define V 5

int minKey(int key[], bool mstSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (!mstSet[v] && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

void printMST(int parent[], int graph[V][V]) {

cout << "Edge Weight\n";

for (int i = 1; i < V; i++)

cout << parent[i] << " - " << i << " " << graph[i][parent[i]] << endl;

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V];

bool mstSet[V]; // included in MST?

for (int i = 0; i < V; i++)

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1; // root of MST

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

printMST(parent, graph);

}

int main() {

int graph[V][V] = {

{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}

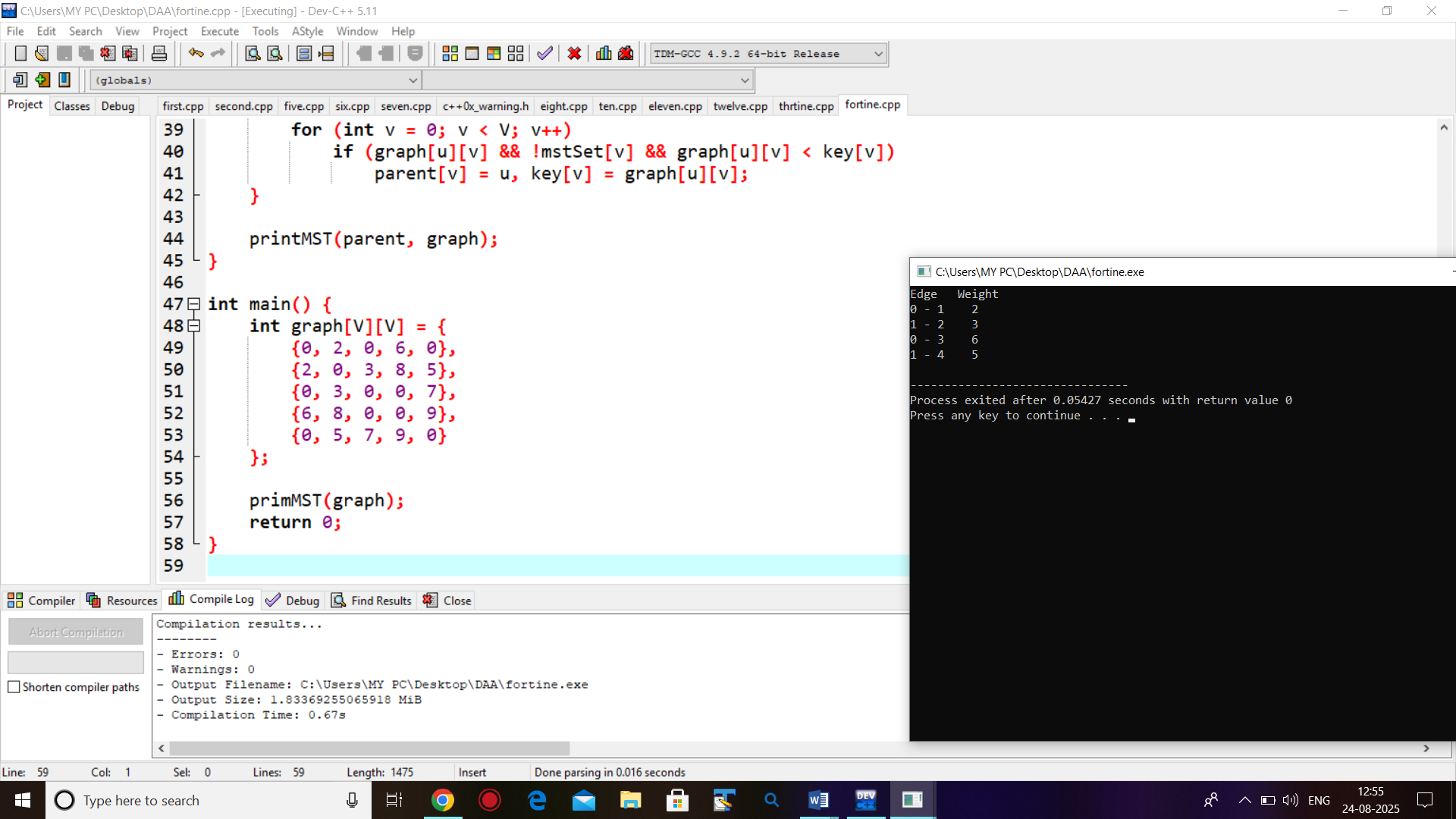
};

primMST(graph);

return 0;

}

**Output:**



**16. Write a program to find Minimum Spanning tree using Kruskal’s algorithm.**

#include <iostream>

using namespace std;

#define V 4

#define E 5

struct Edge {

int u, v, w;

};

int findParent(int parent[], int i) {

if (parent[i] == i) return i;

return findParent(parent, parent[i]);

}

void unionSet(int parent[], int x, int y) {

parent[x] = y;

}

int main() {

Edge edges[E] = {

{0, 1, 10},

{0, 2, 6},

{0, 3, 5},

{1, 3, 15},

{2, 3, 4}

};

for (int i=0; i<E-1; i++)

for (int j=0; j<E-i-1; j++)

if (edges[j].w > edges[j+1].w)

swap(edges[j], edges[j+1]);

int parent[V];

for (int i=0;i<V;i++) parent[i]=i;

cout << "Edges in MST:\n";

int cost = 0;

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

int u = edges[i].u;

int v = edges[i].v;

int set\_u = findParent(parent, u);

int set\_v = findParent(parent, v);

if (set\_u != set\_v) {

cout << u << " - " << v << " : " << edges[i].w << endl;

cost += edges[i].w;

unionSet(parent, set\_u, set\_v);

}

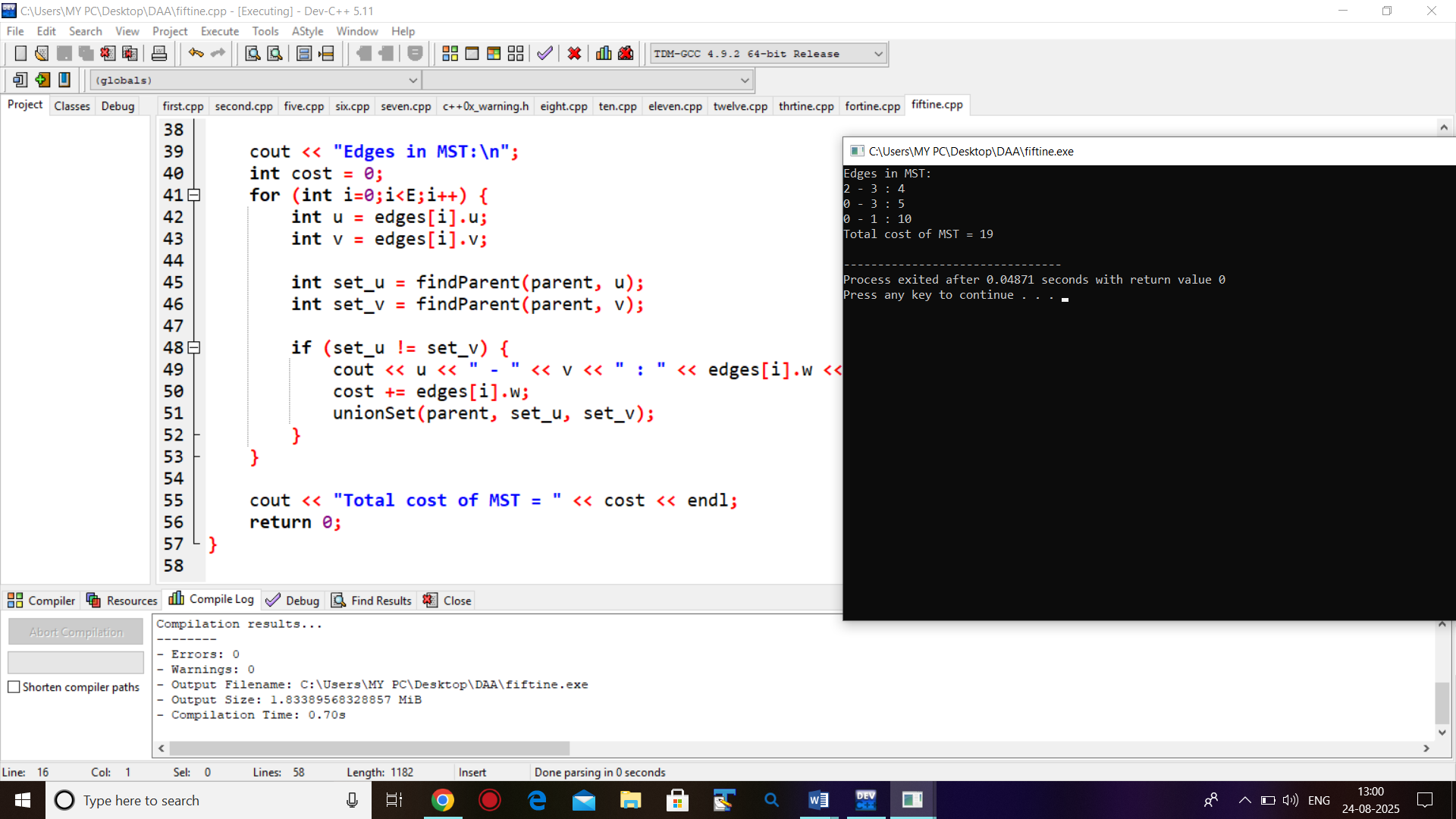
}

cout << "Total cost of MST = " << cost << endl;

return 0;

}

**Output:**



**17. Write a program to find Single Source Shortest Path using Dijkstra’salgorithm.**

#include <iostream>

#include <climits>

using namespace std;

#define V 5

int minDistance(int dist[], bool sptSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (!sptSet[v] && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[]) {

cout << "Vertex \t Distance from Source\n";

for (int i = 0; i < V; i++)

cout << i << "\t\t" << dist[i] << endl;

}

void dijkstra(int graph[V][V], int src) {

int dist[V];

bool sptSet[V];

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V-1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main() {

int graph[V][V] = {

{0, 10, 0, 0, 5},

{0, 0, 1, 0, 2},

{0, 0, 0, 4, 0},

{7, 0, 6, 0, 0},

{0, 3, 9, 2, 0}

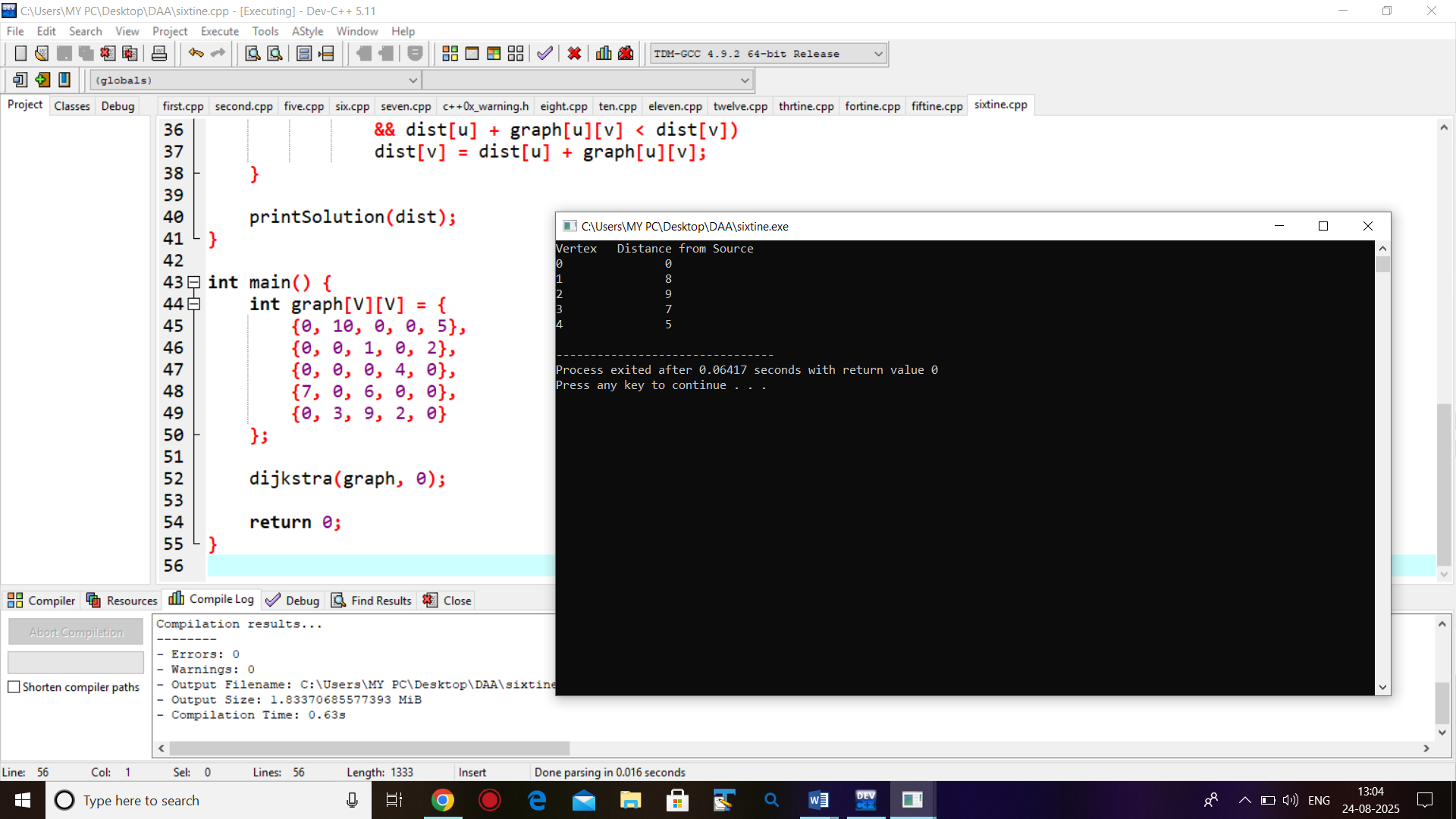
};

dijkstra(graph, 0);

return 0;

}

**Output:**



**18. Write a program to find solution of Knapsack Instance (0/1) .**

#include <iostream>

using namespace std;

int knapsack(int W, int wt[], int val[], int n) {

int dp[n+1][W+1];

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

for (int w=0; w<=W; w++) {

if (i==0 || w==0)

dp[i][w] = 0;

else if (wt[i-1] <= w)

dp[i][w] = max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]);

else

dp[i][w] = dp[i-1][w];

}

}

return dp[n][W];

}

int main() {

int n, W;

cout << "Enter number of items: ";

cin >> n;

cout << "Enter knapsack capacity: ";

cin >> W;

int val[n], wt[n];

cout << "Enter values of items:\n";

for (int i=0; i<n; i++) cin >> val[i];

cout << "Enter weights of items:\n";

for (int i=0; i<n; i++) cin >> wt[i];

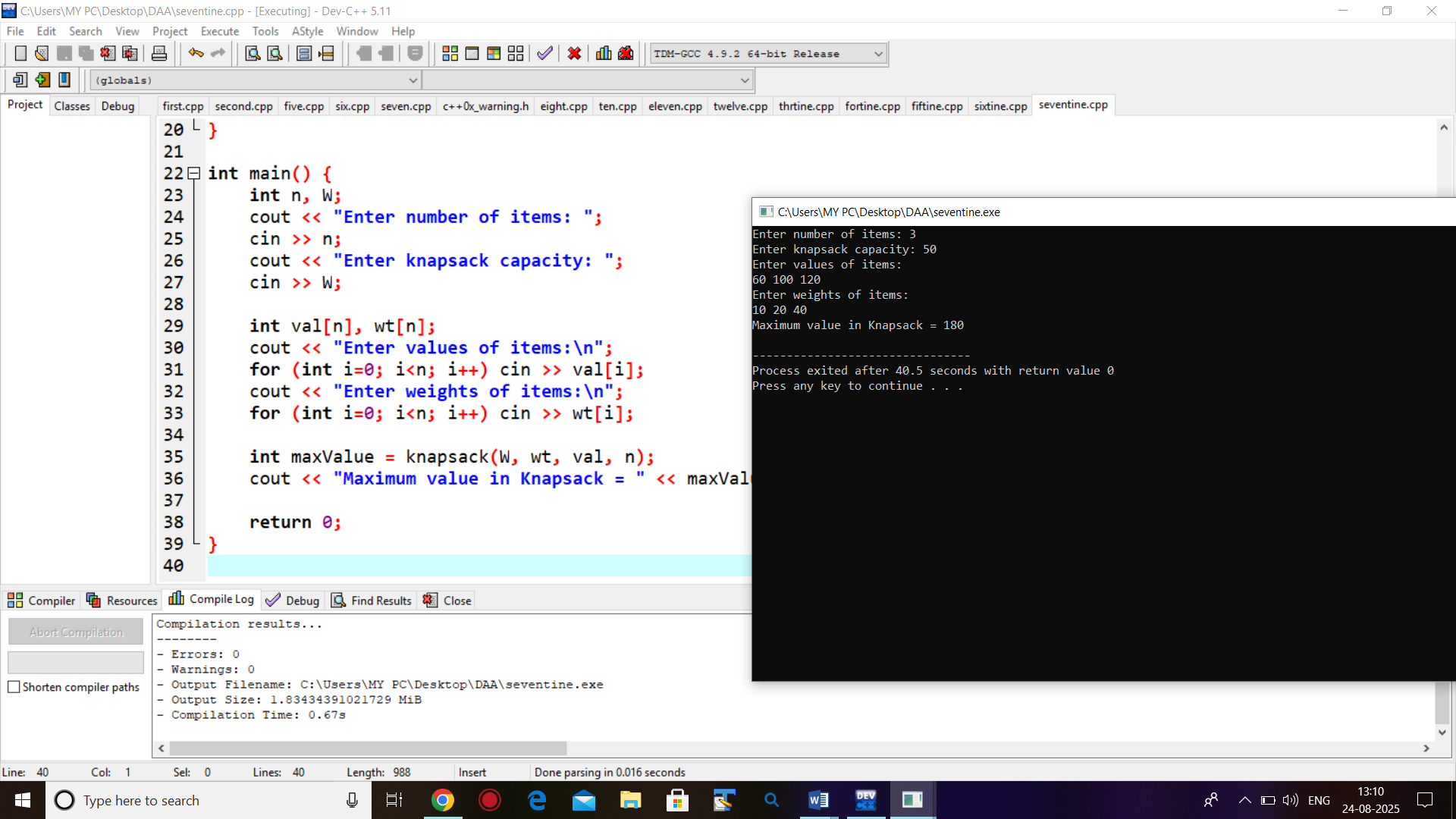
int maxValue = knapsack(W, wt, val, n);

cout << "Maximum value in Knapsack = " << maxValue << endl;

return 0;

}

**Output:**



**19. Write a program to find solution of Matrix Chain Multiplication.**

#include <iostream>

#include <climits>

using namespace std;

int matrixChainOrder(int p[], int n) {

int m[n][n];

for (int i=1; i<n; i++)

m[i][i] = 0;

for (int l=2; l<n; l++) {

for (int i=1; i<n-l+1; i++) {

int j = i+l-1;

m[i][j] = INT\_MAX;

for (int k=i; k<j; k++) {

int q = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];

if (q < m[i][j])

m[i][j] = q;

}

}

}

return m[1][n-1];

}

int main() {

int n;

cout << "Enter number of matrices: ";

cin >> n;

int p[n+1];

cout << "Enter dimensions (p0 p1 ... pn):\n";

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

cin >> p[i];

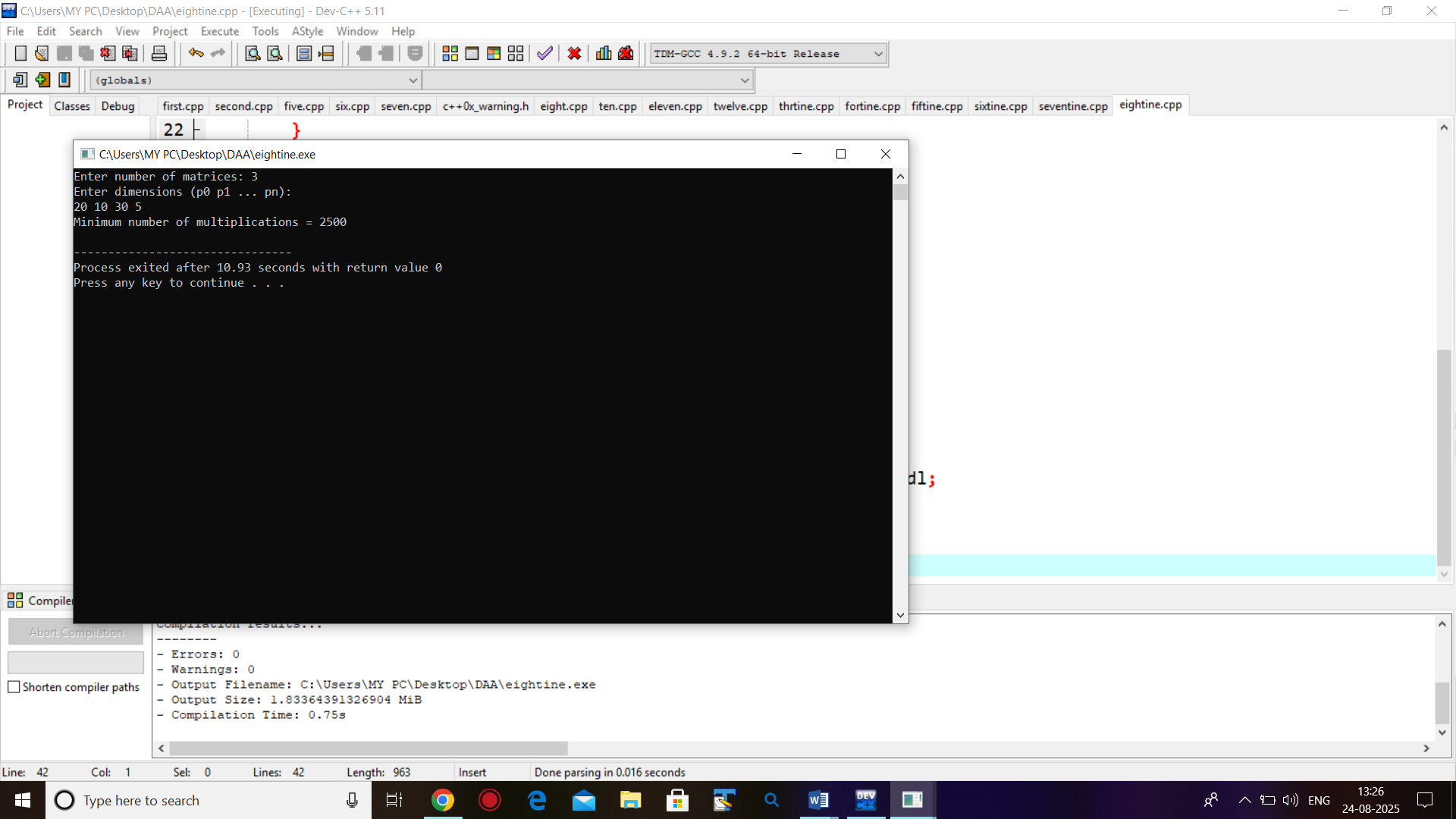
int minCost = matrixChainOrder(p, n+1);

cout << "Minimum number of multiplications = " << minCost << endl;

return 0;

}

**Output:**



**20. Write a program to find shortest path using All Pair Shortest Path algorithm.**

#include <iostream>

using namespace std;

#define INF 9999

int main() {

int V;

cout << "Enter number of vertices: ";

cin >> V;

int dist[V][V];

cout << "Enter adjacency matrix (use " << INF << " for no edge):\n";

for (int i = 0; i < V; i++)

for (int j = 0; j < V; j++)

cin >> dist[i][j];

for (int k = 0; k < V; k++) {

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

for (int j = 0; j < V; j++) {

if (dist[i][k] + dist[k][j] < dist[i][j])

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

cout << "Shortest distances between every pair of vertices:\n";

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

for (int j = 0; j < V; j++) {

if (dist[i][j] == INF)

cout << "INF ";

else

cout << dist[i][j] << " ";

}

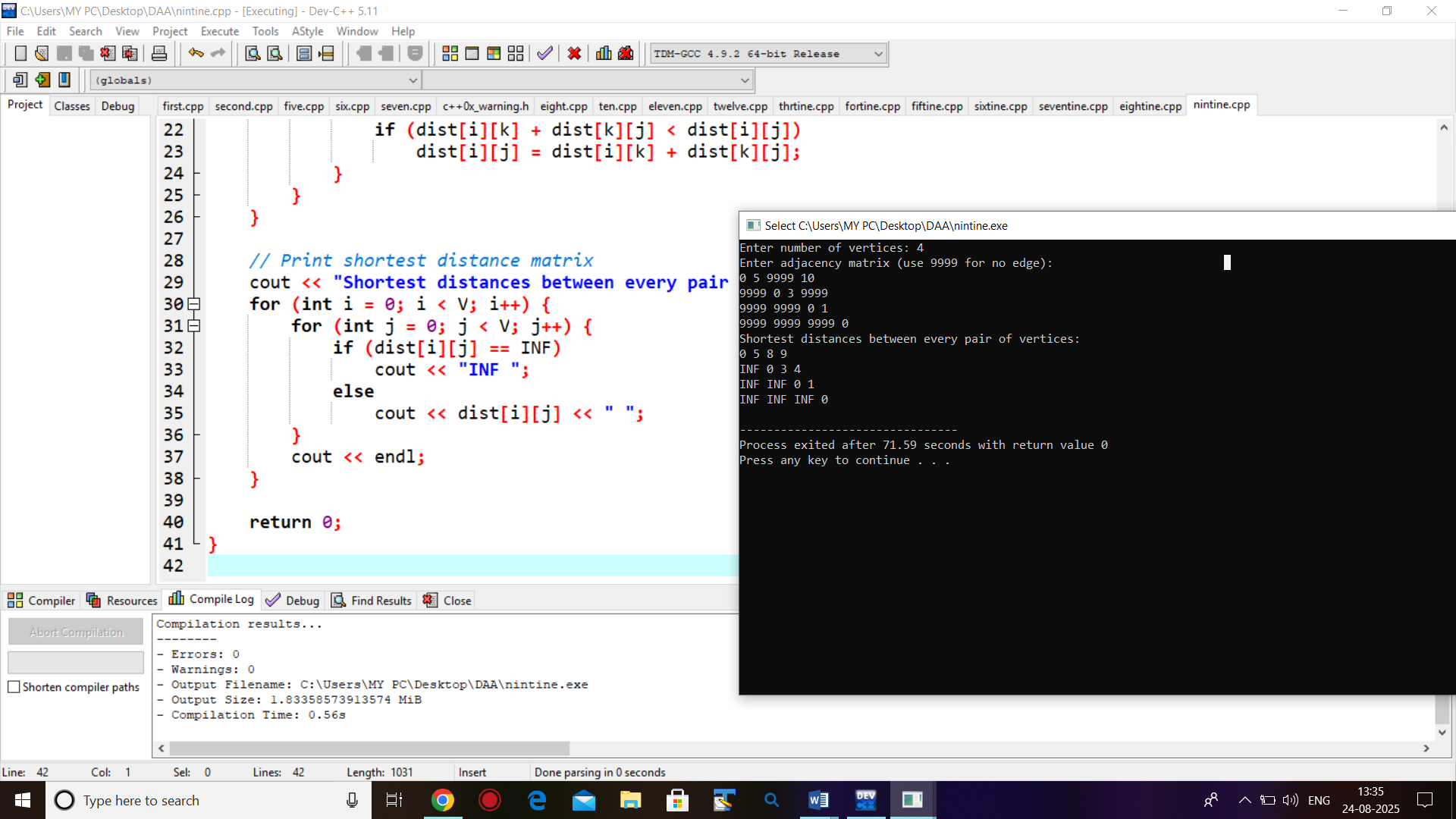
cout << endl;

}

return 0;

}

**Output:**



**21. Write a program to Traverse Graph – Depth First Search, Breadth First Search.**

#include <iostream>

#include <queue>

using namespace std;

const int MAX = 100;

int graph[MAX][MAX];

bool visited[MAX];

int V;

void DFS(int node) {

visited[node] = true;

cout << node << " ";

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

if (graph[node][i] && !visited[i])

DFS(i);

}

}

void BFS(int start) {

bool vis[MAX] = {false};

queue<int> q;

q.push(start);

vis[start] = true;

while (!q.empty()) {

int node = q.front();

q.pop();

cout << node << " ";

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

if (graph[node][i] && !vis[i]) {

q.push(i);

vis[i] = true;

}

}

}

}

int main() {

int E;

cout << "Enter number of vertices: ";

cin >> V;

cout << "Enter number of edges: ";

cin >> E;

for (int i = 0; i < V; i++)

for (int j = 0; j < V; j++)

graph[i][j] = 0;

cout << "Enter edges (u v):\n";

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

int u, v;

cin >> u >> v;

graph[u][v] = 1;

graph[v][u] = 1; // For undirected graph

}

cout << "DFS starting from vertex 0: ";

for (int i = 0; i < V; i++) visited[i] = false;

DFS(0);

cout << "\nBFS starting from vertex 0: ";

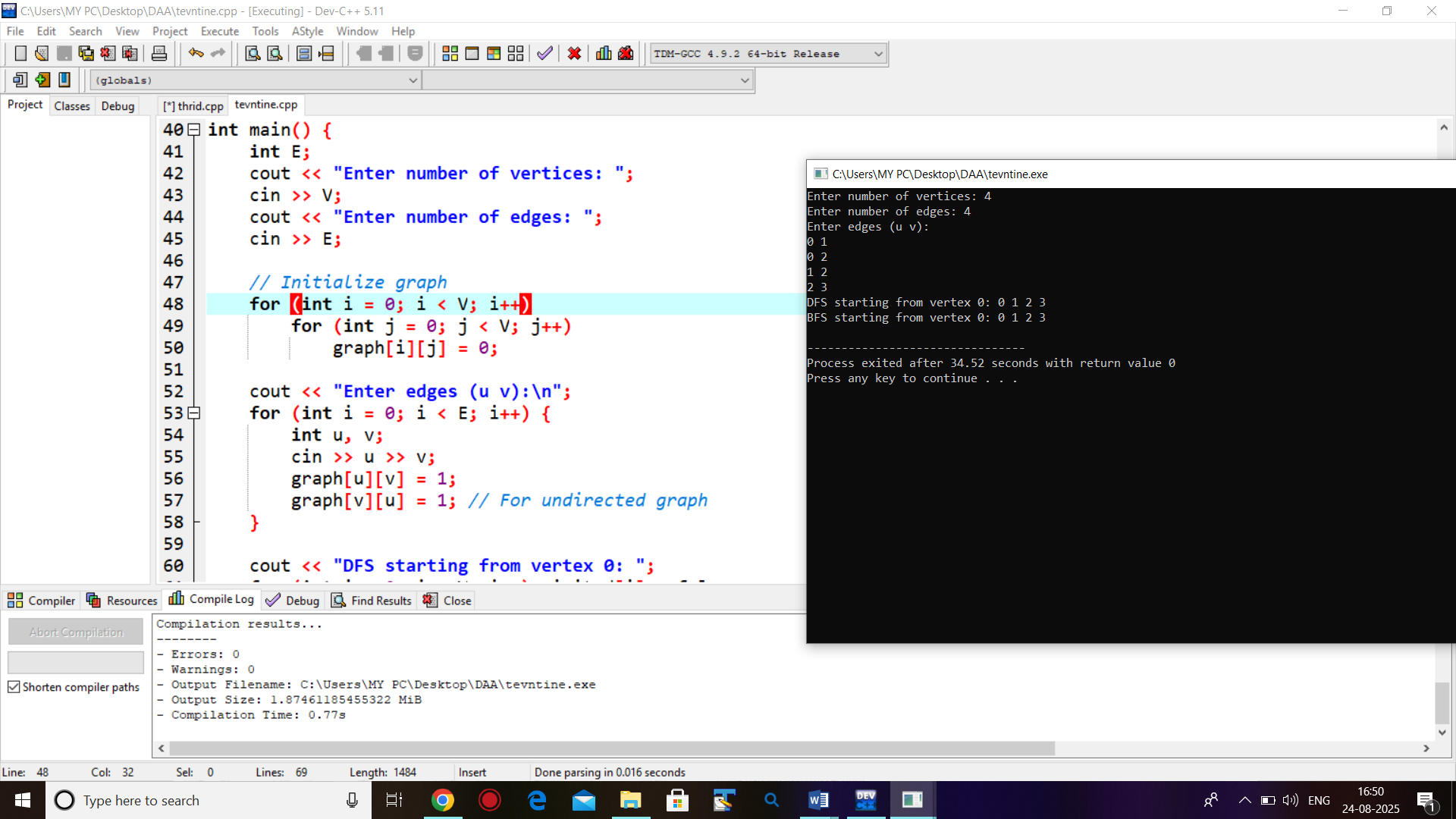
BFS(0);

cout << endl;

return 0;

}

**Output:**



**22. Write a program to find all solutions for N-Queen problem using**

**Backtracking**.

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

void printBoard(const vector<int> &board, int N) {

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

for (int j = 0; j < N; j++) {

if (board[i] == j)

cout << "Q ";

else

cout << ". ";

}

cout << endl;

}

cout << endl;

}

bool isSafe(const vector<int> &board, int row, int col) {

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

// Check same column or diagonal conflicts

if (board[i] == col || abs(board[i] - col) == abs(i - row))

return false;

}

return true;

}

void solveNQueen(int row, int N, vector<int> &board, int &solutionCount) {

if (row == N) {

// All queens are placed successfully

solutionCount++;

printBoard(board, N);

return;

}

for (int col = 0; col < N; col++) {

if (isSafe(board, row, col)) {

board[row] = col;

solveNQueen(row + 1, N, board, solutionCount);

board[row] = -1; }

}

}

int main() {

int N;

cout << "Enter the value of N: ";

cin >> N;

vector<int> board(N, -1);

int solutionCount = 0;

solveNQueen(0, N, board, solutionCount);

cout << "Total solutions: " << solutionCount << endl;

return 0;

}

**Output:**



**23. Write a program for Graph Coloring using backtracking.\**

#include <iostream>

using namespace std;

const int N = 4; // Number of vertices

bool isSafe(int v, int graph[N][N], int color[], int c) {

for (int i = 0; i < N; i++)

if (graph[v][i] && color[i] == c)

return false;

return true;

}

bool graphColoring(int graph[N][N], int m, int color[], int v) {

if (v == N) return true;

for (int c = 1; c <= m; c++) {

if (isSafe(v, graph, color, c)) {

color[v] = c;

if (graphColoring(graph, m, color, v + 1)) return true;

color[v] = 0; // Backtrack

}

}

return false;

}

int main() {

int graph[N][N] = {

{0, 1, 1, 1},

{1, 0, 1, 0},

{1, 1, 0, 1},

{1, 0, 1, 0}

};

int m = 3; // Number of colors

int color[N] = {0};

if (graphColoring(graph, m, color, 0)) {

cout << "Solution exists:\n";

for (int i = 0; i < N; i++)

cout << "Vertex " << i << " -> Color " << color[i] << "\n";

} else {

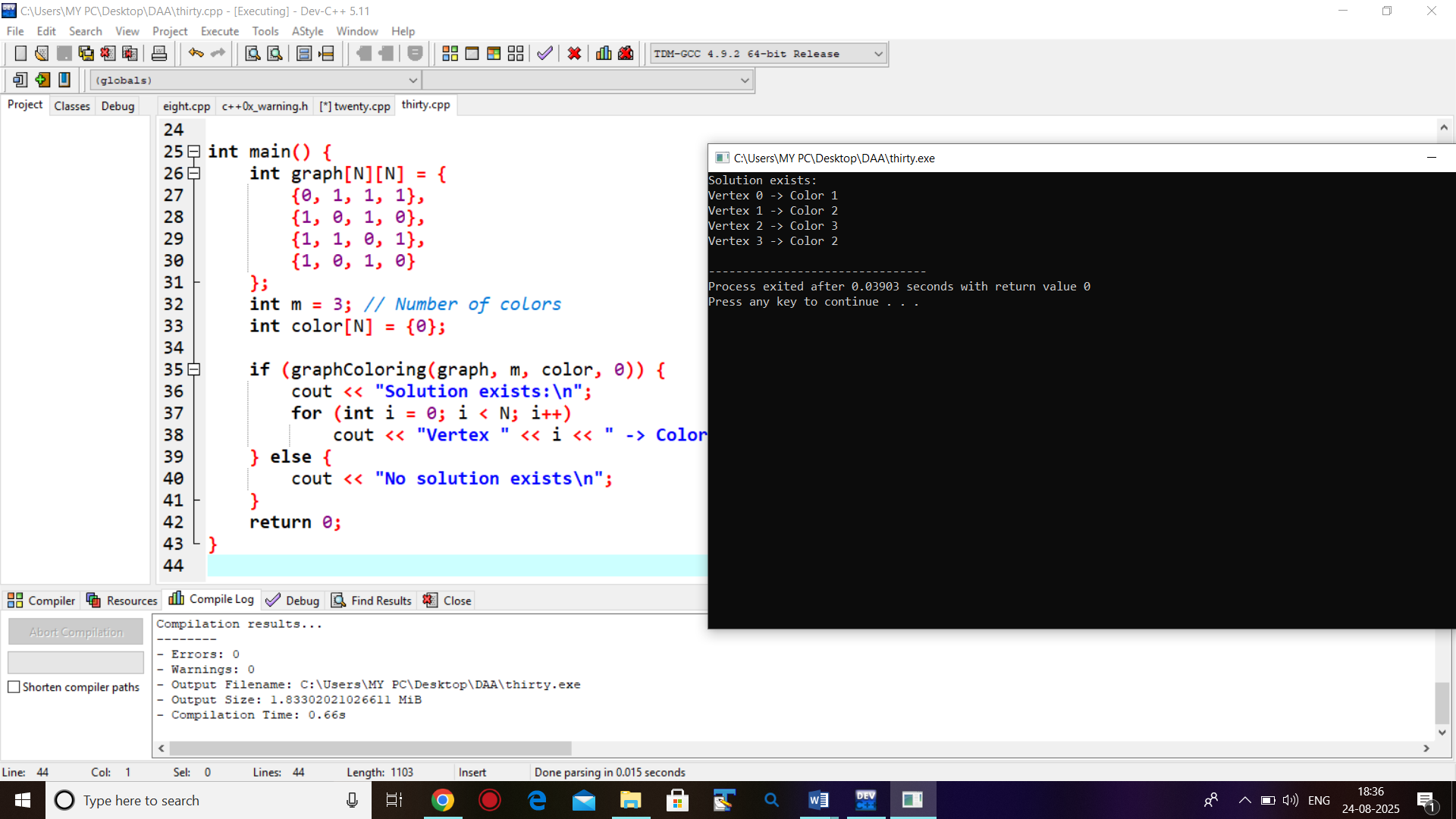
cout << "No solution exists\n";

}

return 0;

}

**Output:**

****