**Algorithms\_Data Structures**

**Exercise 1: Inventory Management System**

**Code:**

**using System;**

**using System.Collections.Generic;**

**class Product {**

**public int ProductId;**

**public string ProductName;**

**public int Quantity;**

**public double Price;**

**}**

**class InventorySystem {**

**Dictionary<int, Product> inventory = new Dictionary<int, Product>();**

**public void AddProduct(Product p) {**

**inventory[p.ProductId] = p;**

**}**

**public void UpdateProduct(Product p) {**

**inventory[p.ProductId] = p;**

**}**

**public void DeleteProduct(int id) {**

**inventory.Remove(id);**

**}**

**public void DisplayAll() {**

**foreach (var p in inventory.Values) {**

**Console.WriteLine($"{p.ProductId} {p.ProductName} {p.Quantity} {p.Price}");**

**}**

**}**

**}**

**class Program {**

**static void Main() {**

**InventorySystem inv = new InventorySystem();**

**inv.AddProduct(new Product { ProductId = 1, ProductName = "Mouse", Quantity = 50, Price = 299.99 });**

**inv.AddProduct(new Product { ProductId = 2, ProductName = "Keyboard", Quantity = 20, Price = 499.49 });**

**inv.UpdateProduct(new Product { ProductId = 1, ProductName = "Mouse", Quantity = 45, Price = 299.99 });**

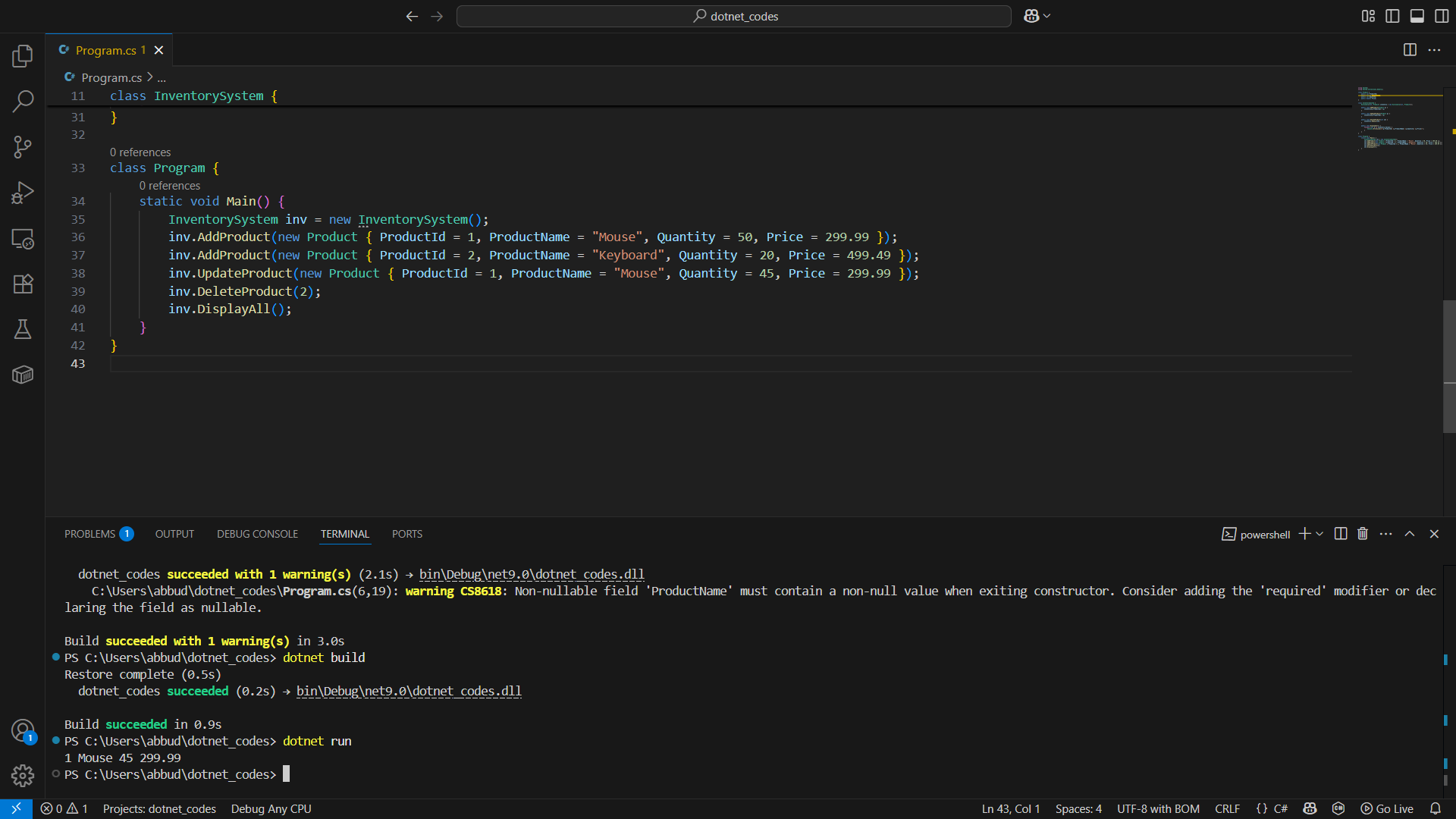
**inv.DeleteProduct(2);**

**inv.DisplayAll();**

**}**

**}**

**Output:**

****

**Exercise 2: E-commerce Platform Search Function**

**Code:**

**using System;**

**class Product {**

**public int ProductId;**

**public string ProductName;**

**public string Category;**

**}**

**class SearchSystem {**

**public static Product LinearSearch(Product[] products, string name) {**

**foreach (var p in products) {**

**if (p.ProductName == name) return p;**

**}**

**return null;**

**}**

**public static Product BinarySearch(Product[] products, string name) {**

**int left = 0, right = products.Length - 1;**

**while (left <= right) {**

**int mid = (left + right) / 2;**

**int cmp = string.Compare(products[mid].ProductName, name);**

**if (cmp == 0) return products[mid];**

**else if (cmp < 0) left = mid + 1;**

**else right = mid - 1;**

**}**

**return null;**

**}**

**public static void Main() {**

**Product[] unsorted = {**

**new Product { ProductId = 1, ProductName = "Mouse", Category = "Electronics" },**

**new Product { ProductId = 2, ProductName = "Keyboard", Category = "Electronics" }**

**};**

**Product[] sorted = {**

**new Product { ProductId = 2, ProductName = "Keyboard", Category = "Electronics" },**

**new Product { ProductId = 1, ProductName = "Mouse", Category = "Electronics" }**

**};**

**var result1 = LinearSearch(unsorted, "Mouse");**

**var result2 = BinarySearch(sorted, "Keyboard");**

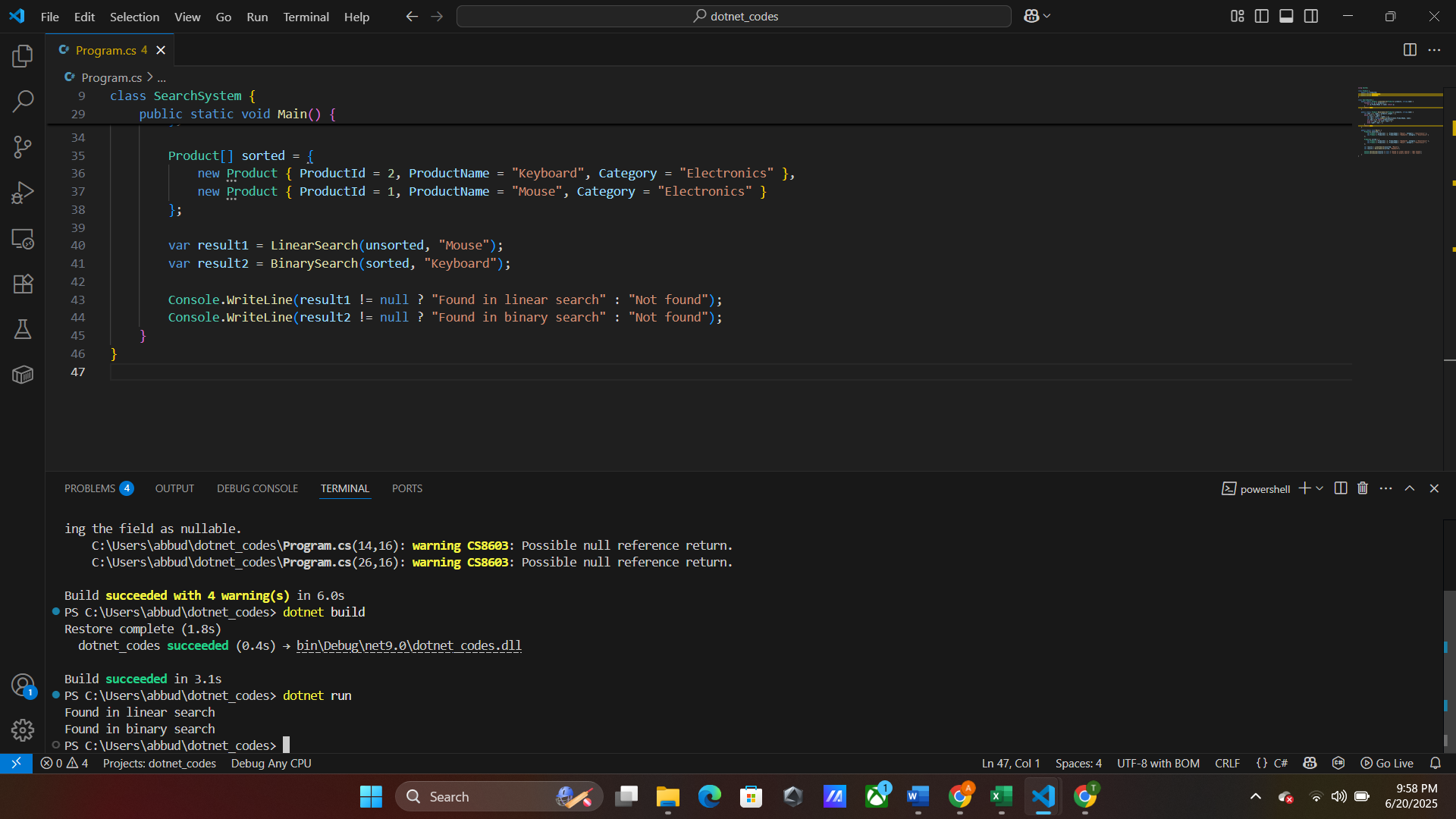
**Console.WriteLine(result1 != null ? "Found in linear search" : "Not found");**

**Console.WriteLine(result2 != null ? "Found in binary search" : "Not found");**

**}**

**}**

**Output:**

****

**Exercise 3: Sorting Customer Orders**

**Code:**

**using System;**

**class Order {**

**public int OrderId;**

**public string CustomerName;**

**public double TotalPrice;**

**}**

**class OrderSorting {**

**public static void BubbleSort(Order[] orders) {**

**int n = orders.Length;**

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

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

**if (orders[j].TotalPrice > orders[j + 1].TotalPrice) {**

**var temp = orders[j];**

**orders[j] = orders[j + 1];**

**orders[j + 1] = temp;**

**}**

**}**

**public static void QuickSort(Order[] orders, int low, int high) {**

**if (low < high) {**

**int pi = Partition(orders, low, high);**

**QuickSort(orders, low, pi - 1);**

**QuickSort(orders, pi + 1, high);**

**}**

**}**

**private static int Partition(Order[] orders, int low, int high) {**

**double pivot = orders[high].TotalPrice;**

**int i = low - 1;**

**for (int j = low; j < high; j++) {**

**if (orders[j].TotalPrice <= pivot) {**

**i++;**

**var temp = orders[i];**

**orders[i] = orders[j];**

**orders[j] = temp;**

**}**

**}**

**var t = orders[i + 1];**

**orders[i + 1] = orders[high];**

**orders[high] = t;**

**return i + 1;**

**}**

**public static void Main() {**

**Order[] orders = {**

**new Order { OrderId = 1, CustomerName = "Alice", TotalPrice = 1500 },**

**new Order { OrderId = 2, CustomerName = "Bob", TotalPrice = 900 }**

**};**

**QuickSort(orders, 0, orders.Length - 1);**

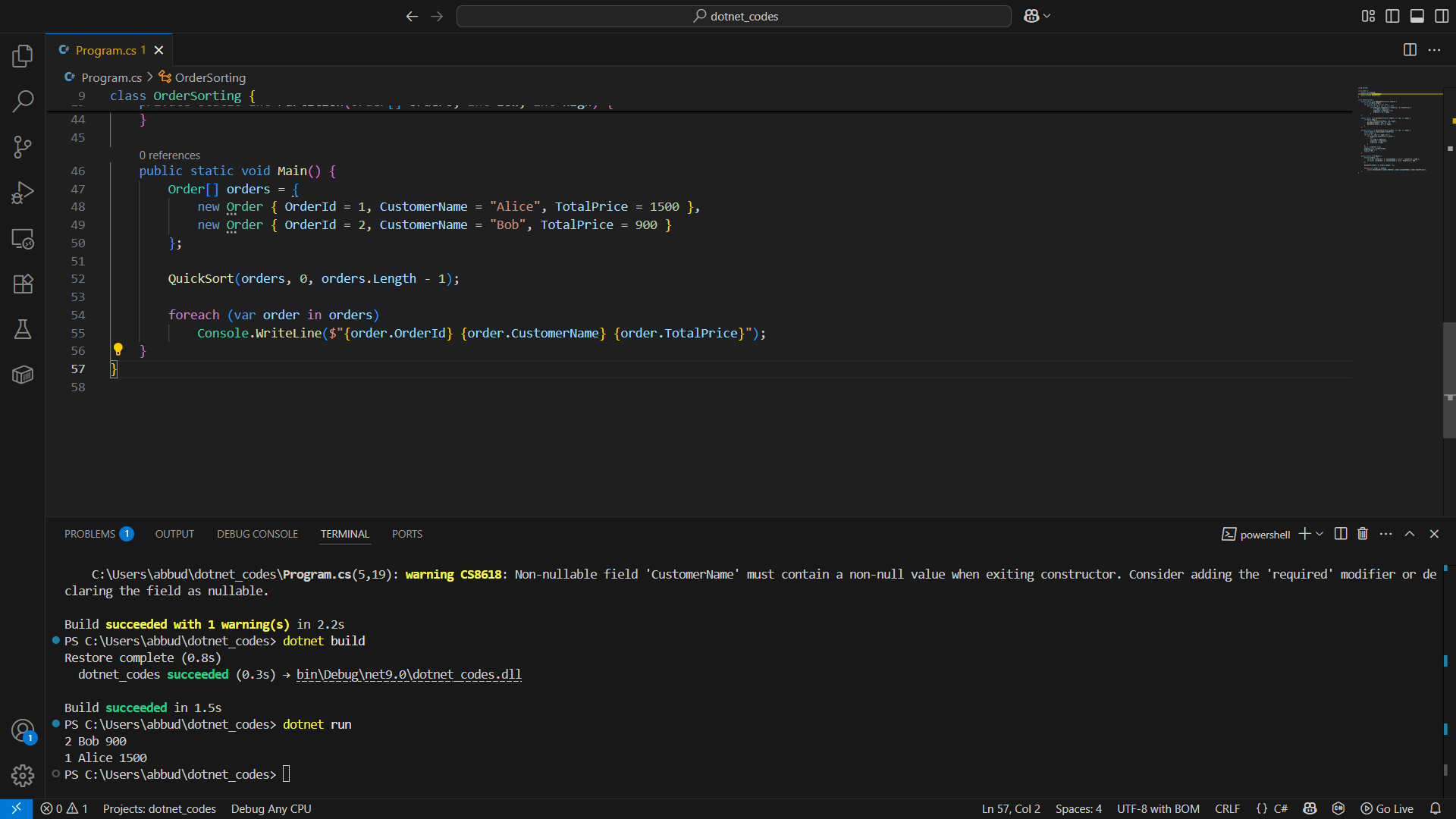
**foreach (var order in orders)**

**Console.WriteLine($"{order.OrderId} {order.CustomerName} {order.TotalPrice}");**

**}**

**}**

**Output:**

****

**Exercise 4: Employee Management System**

**Code:**

**using System;**

**class Employee {**

**public int EmployeeId;**

**public string Name;**

**public string Position;**

**public double Salary;**

**}**

**class EmployeeSystem {**

**Employee[] employees = new Employee[100];**

**int count = 0;**

**public void Add(Employee e) {**

**employees[count++] = e;**

**}**

**public Employee Search(int id) {**

**for (int i = 0; i < count; i++)**

**if (employees[i].EmployeeId == id) return employees[i];**

**return null;**

**}**

**public void Traverse() {**

**for (int i = 0; i < count; i++)**

**Console.WriteLine($"{employees[i].EmployeeId} {employees[i].Name}");**

**}**

**public void Delete(int id) {**

**for (int i = 0; i < count; i++) {**

**if (employees[i].EmployeeId == id) {**

**for (int j = i; j < count - 1; j++)**

**employees[j] = employees[j + 1];**

**count--;**

**break;**

**}**

**}**

**}**

**public static void Main() {**

**EmployeeSystem sys = new EmployeeSystem();**

**sys.Add(new Employee { EmployeeId = 1, Name = "Alice", Position = "Dev", Salary = 50000 });**

**sys.Add(new Employee { EmployeeId = 2, Name = "Bob", Position = "QA", Salary = 40000 });**

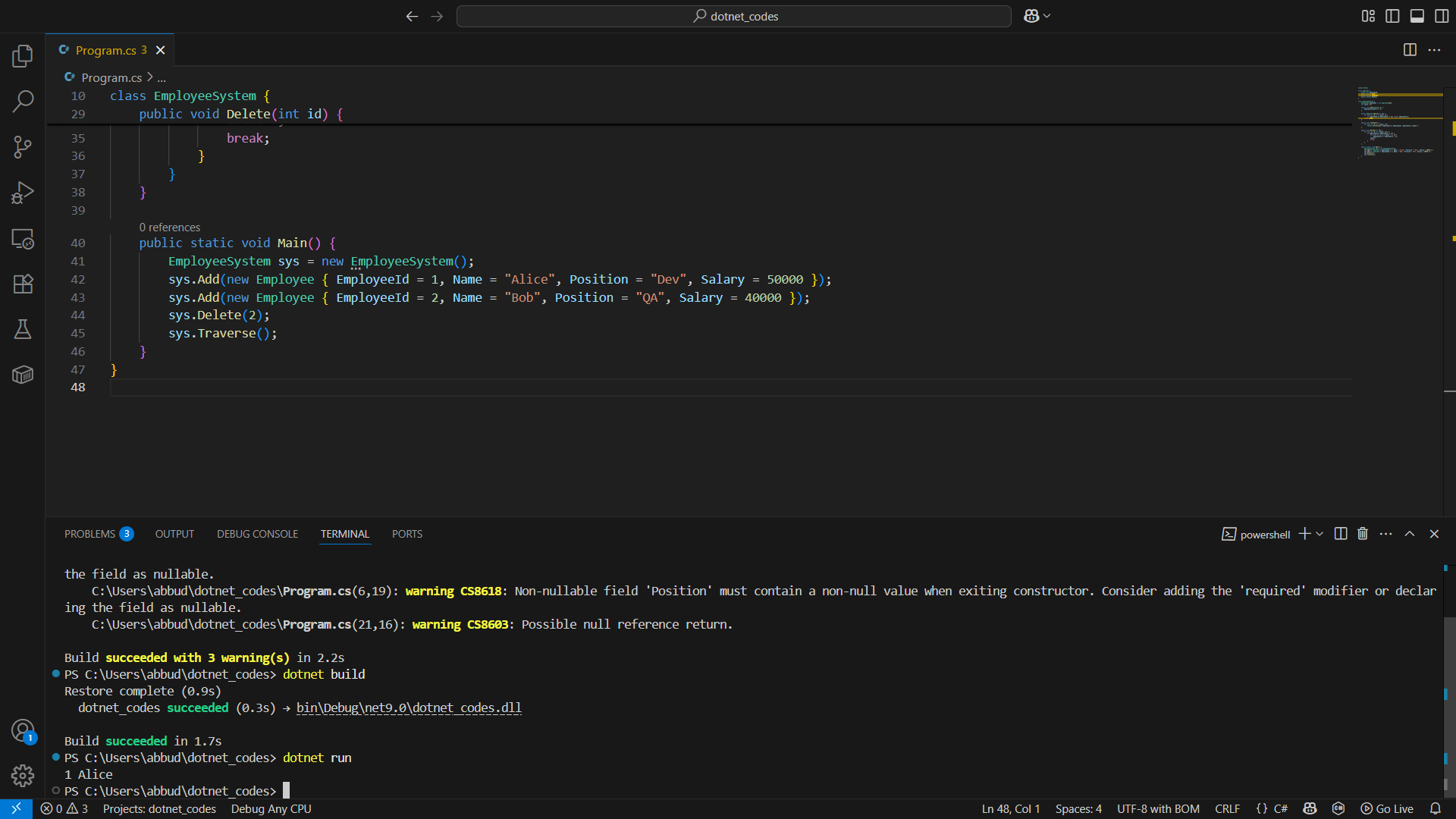
**sys.Delete(2);**

**sys.Traverse();**

**}**

**}**

**Output:**

****

**Exercise 5: Task Management System**

**Code:**

**using System;**

**class Task {**

**public int TaskId;**

**public string TaskName;**

**public string Status;**

**public Task Next;**

**}**

**class TaskList {**

**Task head;**

**public void Add(Task t) {**

**t.Next = head;**

**head = t;**

**}**

**public Task Search(int id) {**

**Task current = head;**

**while (current != null) {**

**if (current.TaskId == id) return current;**

**current = current.Next;**

**}**

**return null;**

**}**

**public void Traverse() {**

**Task current = head;**

**while (current != null) {**

**Console.WriteLine($"{current.TaskId} {current.TaskName} {current.Status}");**

**current = current.Next;**

**}**

**}**

**public void Delete(int id) {**

**Task current = head, prev = null;**

**while (current != null && current.TaskId != id) {**

**prev = current;**

**current = current.Next;**

**}**

**if (current == null) return;**

**if (prev == null) head = head.Next;**

**else prev.Next = current.Next;**

**}**

**public static void Main() {**

**TaskList list = new TaskList();**

**list.Add(new Task { TaskId = 1, TaskName = "Setup", Status = "Pending" });**

**list.Add(new Task { TaskId = 2, TaskName = "Code", Status = "In Progress" });**

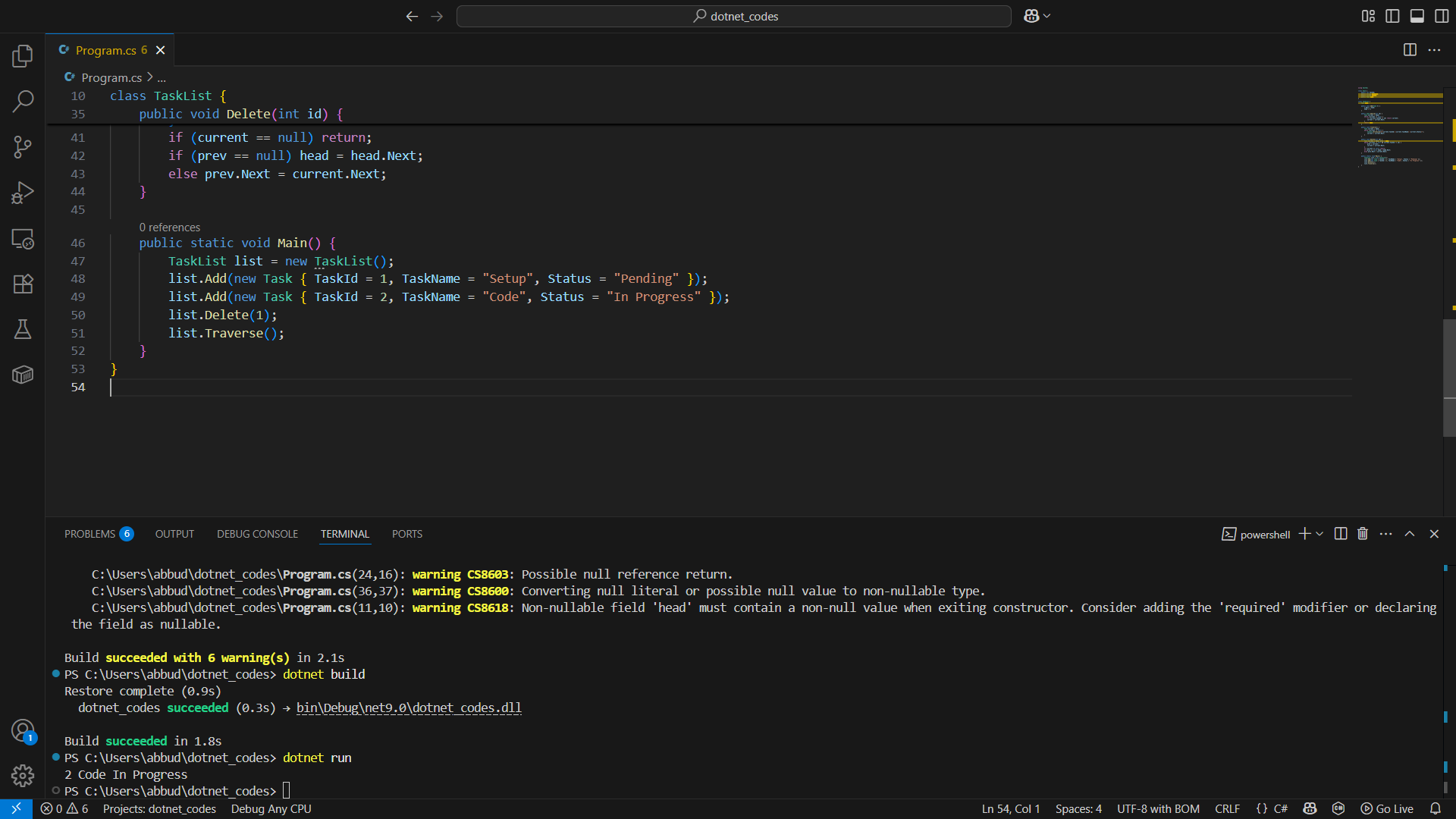
**list.Delete(1);**

**list.Traverse();**

**}**

**}**

**Output:**

****

**Exercise 6: Library Management System**

**Code:**

**using System;**

**class Book {**

**public int BookId;**

**public string Title;**

**public string Author;**

**}**

**class Library {**

**public static Book LinearSearch(Book[] books, string title) {**

**foreach (var b in books)**

**if (b.Title == title) return b;**

**return null;**

**}**

**public static Book BinarySearch(Book[] books, string title) {**

**int l = 0, r = books.Length - 1;**

**while (l <= r) {**

**int m = (l + r) / 2;**

**int cmp = books[m].Title.CompareTo(title);**

**if (cmp == 0) return books[m];**

**else if (cmp < 0) l = m + 1;**

**else r = m - 1;**

**}**

**return null;**

**}**

**public static void Main() {**

**Book[] books = {**

**new Book { BookId = 1, Title = "Alpha", Author = "Author1" },**

**new Book { BookId = 2, Title = "Beta", Author = "Author2" }**

**};**

**var res1 = LinearSearch(books, "Alpha");**

**var res2 = BinarySearch(books, "Beta");**

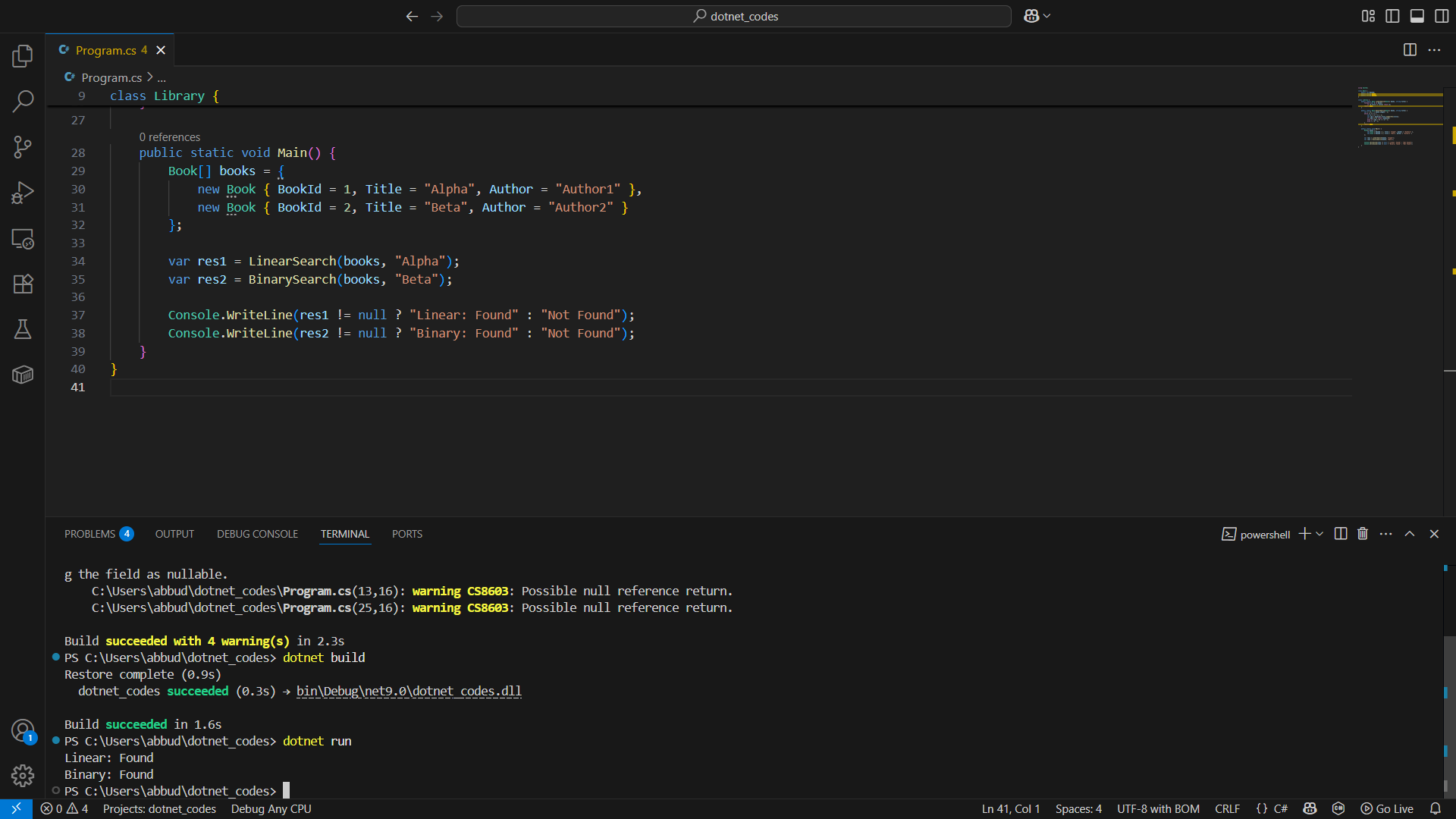
**Console.WriteLine(res1 != null ? "Linear: Found" : "Not Found");**

**Console.WriteLine(res2 != null ? "Binary: Found" : "Not Found");**

**}**

**}**

**Output:**

****

**Exercise 7: Financial Forecasting**

**Code:**

**using System;**

**class Forecast {**

**public static double PredictGrowth(double currentValue, double rate, int years) {**

**if (years == 0) return currentValue;**

**return PredictGrowth(currentValue \* (1 + rate), rate, years - 1);**

**}**

**public static double PredictGrowthMemo(double currentValue, double rate, int years, double[] memo) {**

**if (years == 0) return currentValue;**

**if (memo[years] != 0) return memo[years];**

**return memo[years] = PredictGrowthMemo(currentValue \* (1 + rate), rate, years - 1, memo);**

**}**

**public static void Main() {**

**double result = PredictGrowth(1000, 0.1, 5);**

**Console.WriteLine($"Recursive Result: {result}");**

**double[] memo = new double[6];**

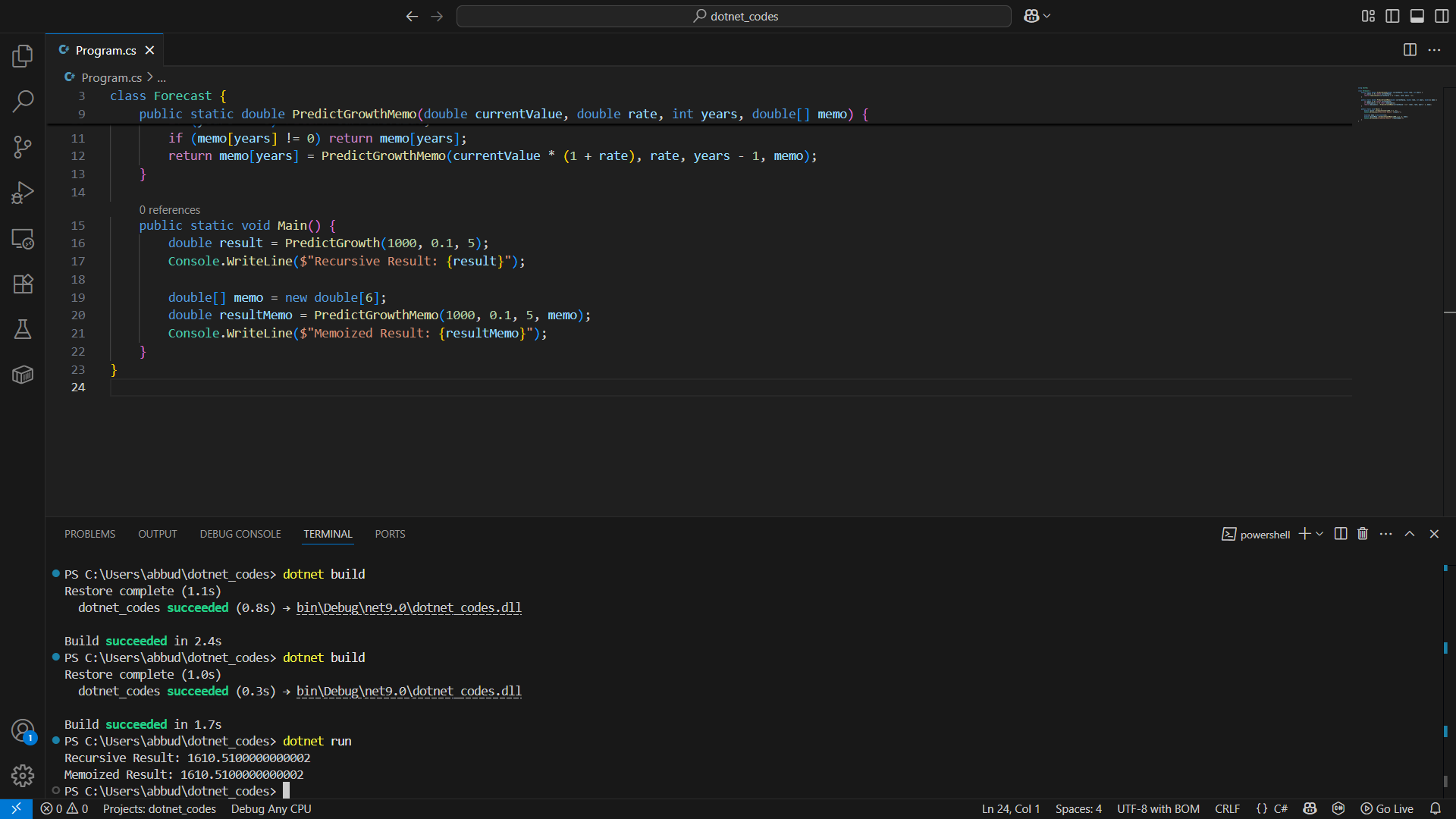
**double resultMemo = PredictGrowthMemo(1000, 0.1, 5, memo);**

**Console.WriteLine($"Memoized Result: {resultMemo}");**

**}**

**}**

**Output:**

****