**Week-1 HandsOn Algorithms Data Structures**

**Exercise 1: Inventory Management System**

**Product.cs**

public class Product

{

public int ProductId { get; set; }

public string ProductName { get; set; }

public int Quantity { get; set; }

public double Price { get; set; }

public Product(int id, string name, int quantity, double price)

{

ProductId = id;

ProductName = name;

Quantity = quantity;

Price = price;

}

public override string ToString()

{

return $"ID: {ProductId}, Name: {ProductName}, Quantity: {Quantity}, Price: ₹{Price}";

}

}

**Inventory.cs**

using System;

using System.Collections.Generic;

public class Inventory

{

private Dictionary<int, Product> products = new Dictionary<int, Product>();

public void AddProduct(Product product)

{

if (products.ContainsKey(product.ProductId))

{

Console.WriteLine("Product with same ID already exists.");

return;

}

products[product.ProductId] = product;

Console.WriteLine("Product added.");

}

public void UpdateProduct(int productId, int quantity, double price)

{

if (products.ContainsKey(productId))

{

products[productId].Quantity = quantity;

products[productId].Price = price;

Console.WriteLine("Product updated.");

}

else

{

Console.WriteLine("Product not found.");

}

}

public void DeleteProduct(int productId)

{

if (products.Remove(productId))

Console.WriteLine("Product deleted.");

else

Console.WriteLine("Product not found.");

}

public void DisplayAll()

{

Console.WriteLine("\nAll Products:");

foreach (var product in products.Values)

{

Console.WriteLine(product);

}

}

}

**Program.cs**

using System;

class Program

{

static void Main(string[] args)

{

Inventory inventory = new Inventory();

inventory.AddProduct(new Product(101, "Mouse", 50, 499.99));

inventory.AddProduct(new Product(102, "Keyboard", 30, 899.50));

inventory.UpdateProduct(101, 45, 459.99);

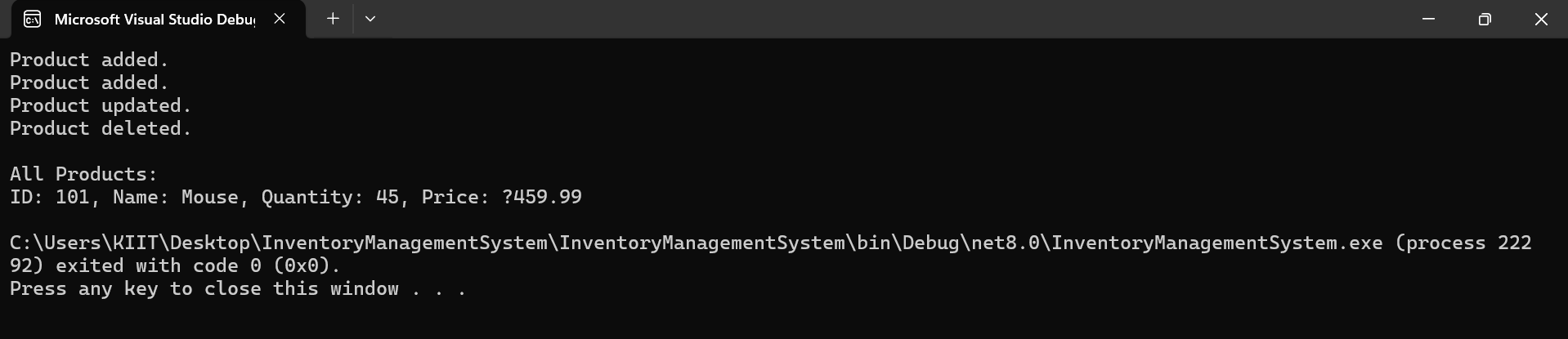
inventory.DeleteProduct(102);

inventory.DisplayAll();

}

}

**ScreenShot of Exercise-1 Output**



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

**Program.cs**

class Program

{

static void Main(string[] args)

{

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shirt", "Apparel"),

new Product(3, "Phone", "Electronics"),

new Product(4, "Shoes", "Footwear"),

new Product(5, "Watch", "Accessories")

};

Console.WriteLine("Linear Search:");

Product result1 = SearchEngine.LinearSearch(products, "Phone");

Console.WriteLine(result1 != null ? result1.ToString() : "Product not found");

Console.WriteLine("\nBinary Search:");

SearchEngine.SortProducts(products);

Product result2 = SearchEngine.BinarySearch(products, "Phone");

Console.WriteLine(result2 != null ? result2.ToString() : "Product not found");

}

}

**SearchEngine.cs**

using System;

public class SearchEngine

{

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

{

foreach (var product in products)

{

if (product.ProductName.Equals(name, StringComparison.OrdinalIgnoreCase))

return product;

}

return null;

}

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

{

int low = 0, high = products.Length - 1;

while (low <= high)

{

int mid = (low + high) / 2;

int comparison = string.Compare(products[mid].ProductName, name, true);

if (comparison == 0)

return products[mid];

else if (comparison < 0)

low = mid + 1;

else

high = mid - 1;

}

return null;

}

public static void SortProducts(Product[] products)

{

Array.Sort(products, (p1, p2) => string.Compare(p1.ProductName, p2.ProductName, true));

}

}

**Product.cs**

public class Product

public class Product

{

public int ProductId { get; set; }

public string ProductName { get; set; }

public string Category { get; set; }

public Product(int id, string name, string category)

{

ProductId = id;

ProductName = name;

Category = category;

}

public override string ToString()

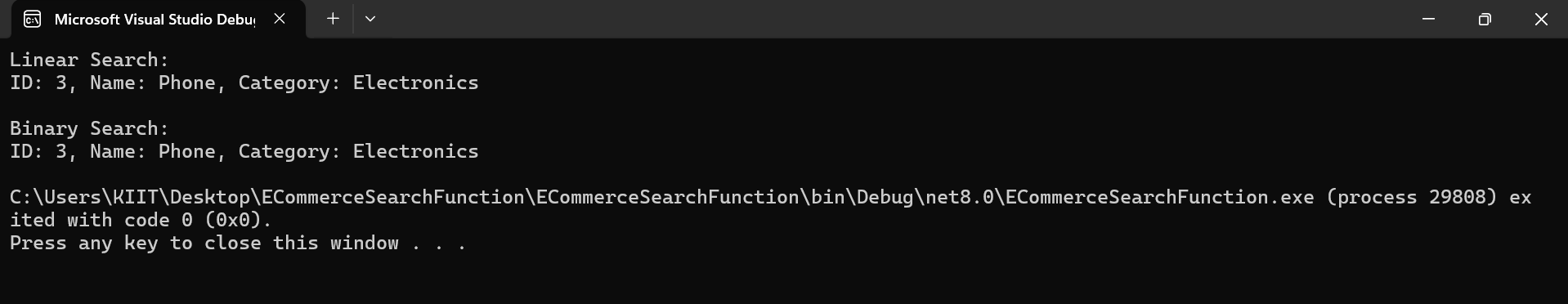
{

return $"ID: {ProductId}, Name: {ProductName}, Category: {Category}";

}

}

**ScreenShot of Exercise-2 Output**



**Exercise 3: Sorting Customer Orders**

**Program.cs**

using System;

class Program

{

static void Main(string[] args)

{

Order[] orders = {

new Order(101, "Alice", 1599.99),

new Order(102, "Bob", 499.00),

new Order(103, "Charlie", 2500.75),

new Order(104, "Diana", 1200.50),

};

Console.WriteLine("Original Orders:");

foreach (var o in orders) Console.WriteLine(o);

Console.WriteLine("\nBubble Sort:");

Sorter.BubbleSort(orders);

foreach (var o in orders) Console.WriteLine(o);

orders = new Order[] {

new Order(101, "Alice", 1599.99),

new Order(102, "Bob", 499.00),

new Order(103, "Charlie", 2500.75),

new Order(104, "Diana", 1200.50),

};

Console.WriteLine("\nQuick Sort:");

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

foreach (var o in orders) Console.WriteLine(o);

}

}

**Sorter.cs**

using System;

public static class Sorter

{

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)

{

(orders[j], orders[j + 1]) = (orders[j + 1], orders[j]);

}

}

}

}

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++;

(orders[i], orders[j]) = (orders[j], orders[i]);

}

}

(orders[i + 1], orders[high]) = (orders[high], orders[i + 1]);

return i + 1;

}

}

**Order.cs**

public class Order

{

public int OrderId { get; set; }

public string CustomerName { get; set; }

public double TotalPrice { get; set; }

public Order(int id, string name, double price)

{

OrderId = id;

CustomerName = name;

TotalPrice = price;

}

public override string ToString()

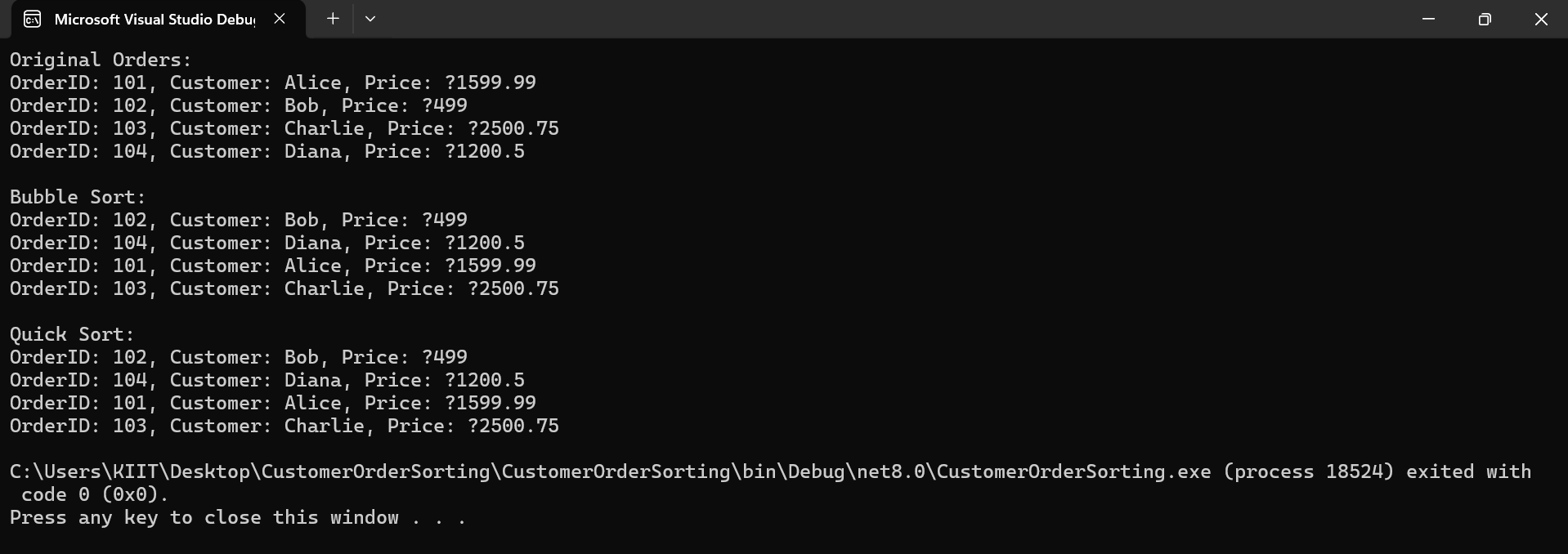
{

return $"OrderID: {OrderId}, Customer: {CustomerName}, Price: ₹{TotalPrice}";

}

}

**ScreenShot of Exercise-3 Output**



**Exercise 4: Employee Management System**

**Progarm.cs**

using System;

class Program

{

static void Main(string[] args)

{

EmployeeManager manager = new EmployeeManager(5);

manager.AddEmployee(new Employee(1, "Alice", "Manager", 60000));

manager.AddEmployee(new Employee(2, "Bob", "Developer", 50000));

manager.AddEmployee(new Employee(3, "Charlie", "Designer", 45000));

Console.WriteLine("\nAll Employees:");

manager.DisplayAll();

Console.WriteLine("\nSearching for Employee with ID 2:");

Employee? emp = manager.SearchEmployee(2);

Console.WriteLine(emp != null ? emp.ToString() : "Not found");

Console.WriteLine("\nDeleting Employee with ID 1:");

manager.DeleteEmployee(1);

Console.WriteLine("\nAll Employees After Deletion:");

manager.DisplayAll();

}

}

**Employee.cs**

public class Employee

{

public int EmployeeId { get; set; }

public string Name { get; set; }

public string Position { get; set; }

public double Salary { get; set; }

public Employee(int id, string name, string position, double salary)

{

EmployeeId = id;

Name = name;

Position = position;

Salary = salary;

}

public override string ToString()

{

return $"ID: {EmployeeId}, Name: {Name}, Position: {Position}, Salary: ₹{Salary}";

}

}

**EmployeeManager.cs**

using System;

public class EmployeeManager

{ private Employee[] employees;

private int count = 0;

public EmployeeManager(int capacity)

{ employees = new Employee[capacity];

}

public void AddEmployee(Employee emp)

{

if (count < employees.Length)

{

employees[count++] = emp;

Console.WriteLine("Employee added.");

}

else

{ Console.WriteLine("Array is full. Cannot add more employees.");

}

}

public Employee SearchEmployee(int id)

{

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

{

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

return employees[i];

}

return null;

}

public void DisplayAll()

{

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

{

Console.WriteLine(employees[i]);

}

}

public void DeleteEmployee(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];

}

employees[--count] = null;

Console.WriteLine("Employee deleted.");

return;

}

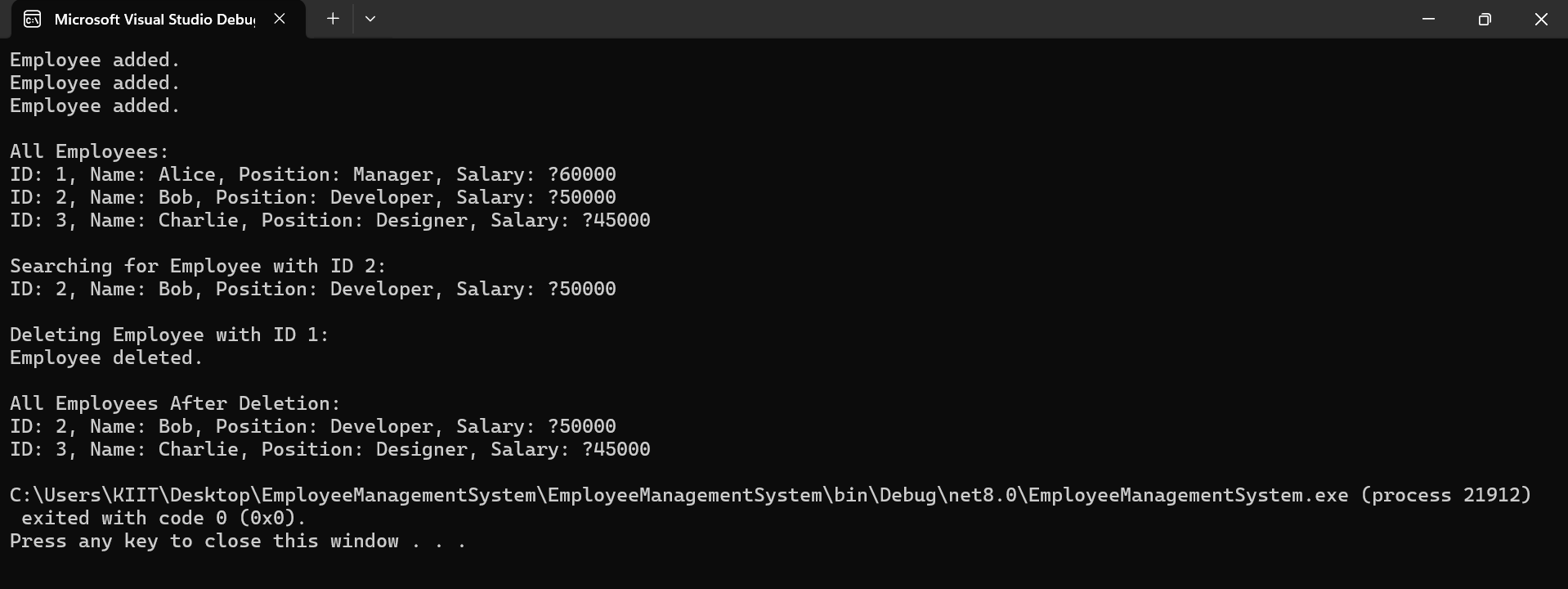
}

Console.WriteLine("Employee not found.");

}

}

**ScreenShot of Exercise-4 Output**



**Exercise 5: Task Management System**

public void Traverse()

{

TaskNode? current = head;

while (current != null)

{

Console.WriteLine(current.Data);

current = current.Next;

}

}

}

**TaskList.cs**

using System;

public class TaskList

{ private TaskNode? head = null;

public void AddTask(Task task)

{ TaskNode newNode = new TaskNode(task);

if (head == null)

{ head = newNode;

}

else

{ TaskNode current = head;

while (current.Next != null)

current = current.Next;

current.Next = newNode;

}

Console.WriteLine("Task added.");

}

public Task? SearchTask(int taskId)

{

TaskNode? current = head;

while (current != null)

{

if (current.Data.TaskId == taskId)

return current.Data;

current = current.Next;

}

return null;

}

public void DeleteTask(int taskId)

{

if (head == null)

{ Console.WriteLine("No tasks to delete.");

return;

}

if (head.Data.TaskId == taskId)

{

head = head.Next;

Console.WriteLine("Task deleted.");

return;

}

TaskNode? current = head;

while (current.Next != null && current.Next.Data.TaskId != taskId)

{ current = current.Next;

}

if (current.Next != null)

{ current.Next = current.Next.Next;

Console.WriteLine("Task deleted.");

}

else

{ Console.WriteLine("Task not found.");

}

} continue……

**TaskNode.cs**

public class TaskNode

{

public Task Data { get; set; }

public TaskNode? Next { get; set; }

public TaskNode(Task task)

{

Data = task;

Next = null;

}

}

**Task.cs**

public class Task

{

public int TaskId { get; set; }

public string TaskName { get; set; }

public string Status { get; set; }

public Task(int id, string name, string status)

{

TaskId = id;

TaskName = name;

Status = status;

}

public override string ToString()

{

return $"ID: {TaskId}, Task: {TaskName}, Status: {Status}";

}

}

**Program.cs**

using System;

class Program

{ static void Main(string[] args)

{ TaskList taskList = new TaskList();

taskList.AddTask(new Task(1, "Design UI", "Pending"));

taskList.AddTask(new Task(2, "Develop API", "In Progress"));

taskList.AddTask(new Task(3, "Test Functionality", "Pending"));

Console.WriteLine("\nAll Tasks:");

taskList.Traverse();

Console.WriteLine("\nSearching for Task ID 2:");

Task? task = taskList.SearchTask(2);

Console.WriteLine(task != null ? task.ToString() : "Task not found.");

Console.WriteLine("\nDeleting Task ID 1:");

taskList.DeleteTask(1);

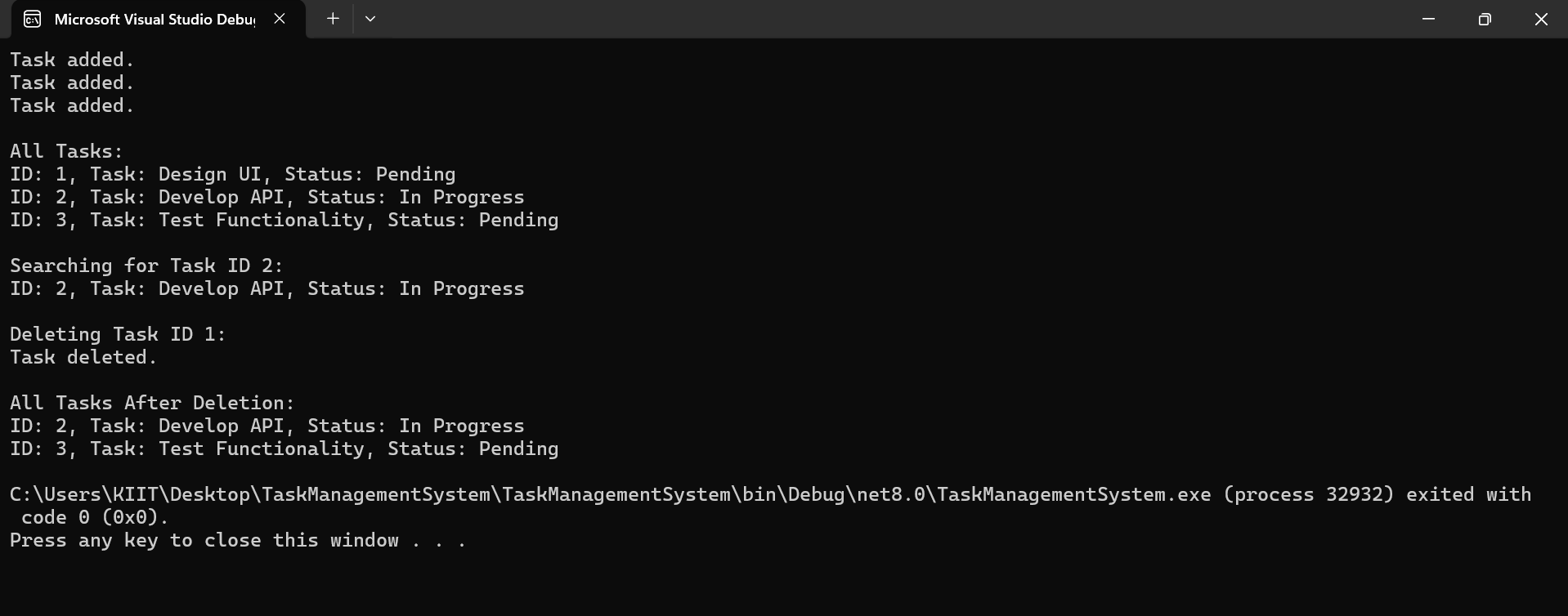
Console.WriteLine("\nAll Tasks After Deletion:");

taskList.Traverse();

}

}

**ScreenShot of Exercise-5 Output**



**Exercise 6: Library Management System**

**Program.cs**

using System;

class Program

{

static void Main(string[] args)

{

Book[] books = {

new Book(1, "C Programming", "Dennis Ritchie"),

new Book(2, "Clean Code", "Robert Martin"),

new Book(3, "Algorithms", "Cormen"),

new Book(4, "Design Patterns", "Gamma"),

new Book(5, "Data Structures", "Mark Allen Weiss")

};

Console.WriteLine("🔎 Linear Search for 'Clean Code':");

Book? result1 = Library.LinearSearch(books, "Clean Code");

Console.WriteLine(result1 != null ? result1.ToString() : "Book not found.");

Console.WriteLine("\n Sorting books for Binary Search...");

Library.SortBooksByTitle(books);

Console.WriteLine("\n Binary Search for 'Clean Code':");

Book? result2 = Library.BinarySearch(books, "Clean Code");

Console.WriteLine(result2 != null ? result2.ToString() : "Book not found.");

}

}

**Library.cs**

using System;

public static class Library

{

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

{

foreach (var book in books)

{

if (book.Title.Equals(title, StringComparison.OrdinalIgnoreCase))

return book;

}

return null;

}

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

{

int low = 0, high = books.Length - 1;

while (low <= high)

{

int mid = (low + high) / 2;

int cmp = string.Compare(books[mid].Title, title, true);

if (cmp == 0) return books[mid];

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

else high = mid - 1;

}

return null;

}

public static void SortBooksByTitle(Book[] books)

{

Array.Sort(books, (a, b) => string.Compare(a.Title, b.Title, true));

}

}

**Book.cs**

public class Book

{

public int BookId { get; set; }

public string Title { get; set; }

public string Author { get; set; }

public Book(int id, string title, string author)

{

BookId = id;

Title = title;

Author = author;

}

public override string ToString()

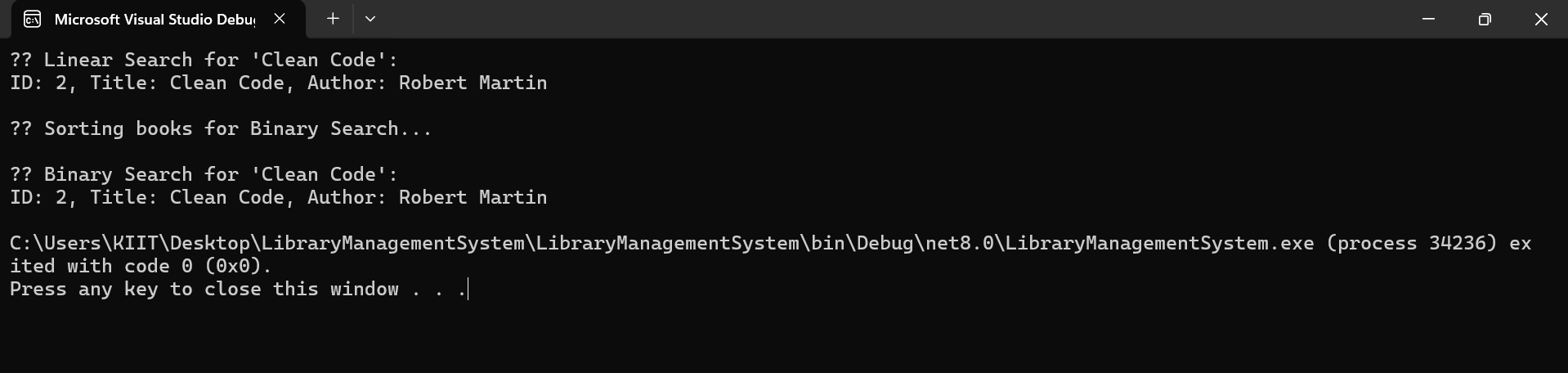
{

return $"ID: {BookId}, Title: {Title}, Author: {Author}";

}

}

**ScreenShot of Exercise-6 Output**



**Exercise 7: Financial Forecasting**

**ScreenShot of Exercise-7 Output**



**Program.cs**

using System;

class Program

{

static void Main(string[] args)

{

double presentValue = 10000;

double growthRate = 0.05;

int years = 5;

Console.WriteLine(" Financial Forecasting using Recursion");

double futureValue = Forecaster.PredictFutureValue(presentValue, growthRate, years);

Console.WriteLine($"Future value (recursive): {futureValue:F2}");

double futureValueIterative = Forecaster.PredictFutureValueIterative(presentValue, growthRate, years);

Console.WriteLine($"Future value (iterative): {futureValueIterative:F2}");

}

}

**Forecaster.cs**

using System;

public class Forecaster

{ public static double PredictFutureValue(double presentValue, double growthRate, int years)

{

if (years == 0)

return presentValue;

return PredictFutureValue(presentValue \* (1 + growthRate), growthRate, years - 1);

}

public static double PredictFutureValueIterative(double presentValue, double growthRate, int years)

{

double result = presentValue;

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

{

result \*= (1 + growthRate);

}

return result;

}

}