**Task-2**

**Problem 1:** Create a list of objects representing Books with the following properties: BookID (int),Title (string), Author (string), Price (decimal). Perform the following tasks using LINQ: Filters to find books authored by "John Doe" with a price between 200 and 800.

**Ans:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

class Book

{

public int BookID { get; set; }

public string Title { get; set; }

public string Author { get; set; }

public decimal Price { get; set; }

}

namespace Problem\_1

{

class Program

{

static void Main(string[] args)

{

List<Book> books = new List<Book>

{

new Book { BookID = 1, Title = "C# Basics", Author = "John Doe", Price = 500 },

new Book { BookID = 2, Title = "Advanced C#", Author = "Jane Smith", Price = 900 },

new Book { BookID = 3, Title = "LINQ in Action", Author = "John Doe", Price = 750 },

new Book { BookID = 4, Title = "Python 101", Author = "Mark Lee", Price = 300 },

new Book { BookID = 5, Title = "Data Structures", Author = "John Doe", Price = 180 }

};

var filteredBooks = books

.Where(book => book.Author == "John Doe" && book.Price >= 200 && book.Price <= 800)

.ToList();

Console.WriteLine("Books authored by 'John Doe' with price between 200 and 800:");

foreach (var book in filteredBooks)

{

Console.WriteLine($"ID: {book.BookID}, Title: {book.Title}, Price: {book.Price}");

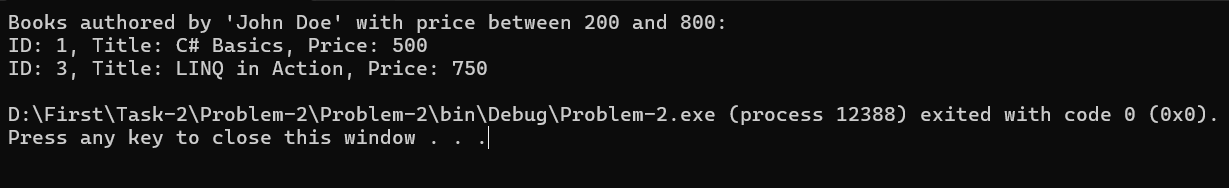
}

}

}

}

**Output:**



**Problem 2:** Create a list of employees with the following properties: EmployeeID (int) Name (string) Department (string) Salary (decimal)

Task: Use LINQ to filter employees from the "IT" department who have a salary between 50,000 and 100,000.

**Ans:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

class Employee

{

public int EmployeeID { get; set; }

public string Name { get; set; }

public string Department { get; set; }

public decimal Salary { get; set; }

}

namespace Problem\_2

{

class Program

{

static void Main(string[] args)

{

List<Employee> employees = new List<Employee>

{

new Employee { EmployeeID = 1, Name = "Tonmoy", Department = "IT", Salary = 60000 },

new Employee { EmployeeID = 2, Name = "Turjo", Department = "HR", Salary = 55000 },

new Employee { EmployeeID = 3, Name = "Hasan", Department = "IT", Salary = 80000 },

new Employee { EmployeeID = 4, Name = "Moho", Department = "Finance", Salary = 75000 },

new Employee { EmployeeID = 5, Name = "Hasan", Department = "IT", Salary = 45000 },

new Employee { EmployeeID = 6, Name = "Amit", Department = "IT", Salary = 95000 }

};

var filteredEmployees = employees

.Where(emp => emp.Department == "IT" && emp.Salary >= 50000 && emp.Salary <= 100000)

.ToList();

Console.WriteLine("Filtered Employees (IT Department, Salary between 50000 - 100000):\n");

foreach (var emp in filteredEmployees)

{

Console.WriteLine($"ID: {emp.EmployeeID}, Name: {emp.Name}, Department: {emp.Department}, Salary: {emp.Salary}");

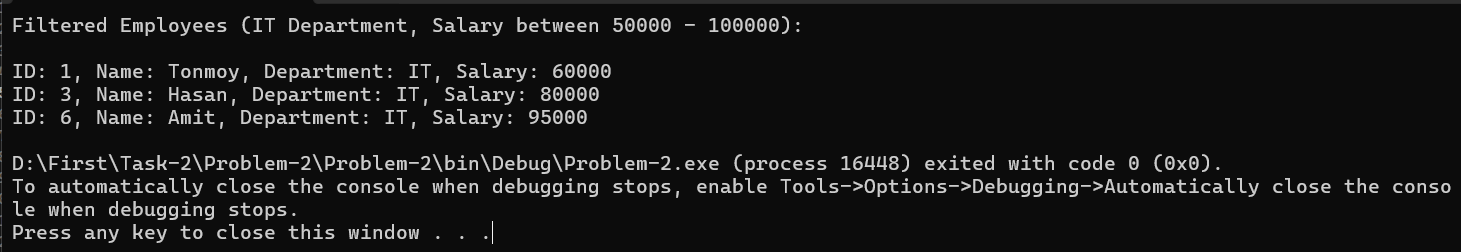
}

}

}

}

**Output:**



**Problem 3:**

Scenario: SpaceX rockets have multiple stages, each serving a different purpose during a launch. You need to create an enumeration that represents different rocket stages (PreLaunch, Liftoff, StageSeparation, OrbitInsertion, Reentry, Landing).

Problem Statement:

 Define an enum RocketStage representing the different stages of a rocket launch.

 Write a Rocket class that has a method AdvanceStage() to transition to the next stage.

 Implement a program that simulates a SpaceX rocket launch by advancing through all

stages and printing the current stage.

Output:

Preparing for launch. Liftoff! Rocket is ascending. Stage separation successful. Rocket has reached orbit.

Reentering Earth's atmosphere. Rocket has landed successfully.

**Ans:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

enum RocketStage

{

PreLaunch,

Liftoff,

StageSeparation,

OrbitInsertion,

Reentry,

Landing

}

class Rocket

{

private RocketStage currentStage;

public Rocket()

{

currentStage = RocketStage.PreLaunch;

}

public void AdvanceStage()

{

switch (currentStage)

{

case RocketStage.PreLaunch:

Console.WriteLine("Preparing for launch.");

break;

case RocketStage.Liftoff:

Console.WriteLine("Liftoff! Rocket is ascending.");

break;

case RocketStage.StageSeparation:

Console.WriteLine("Stage separation successful.");

break;

case RocketStage.OrbitInsertion:

Console.WriteLine("Rocket has reached orbit.");

break;

case RocketStage.Reentry:

Console.WriteLine("Reentering Earth's atmosphere.");

break;

case RocketStage.Landing:

Console.WriteLine("Rocket has landed successfully.");

break;

}

if (currentStage < RocketStage.Landing)

{

currentStage++;

}

}

public bool IsMissionComplete()

{

return currentStage == RocketStage.Landing;

}

}

namespace Problem\_3

{

class Program

{

static void Main(string[] args)

{

Rocket rocket = new Rocket();

while (true)

{

rocket.AdvanceStage();

if (rocket.IsMissionComplete())

break;

System.Threading.Thread.Sleep(1000);

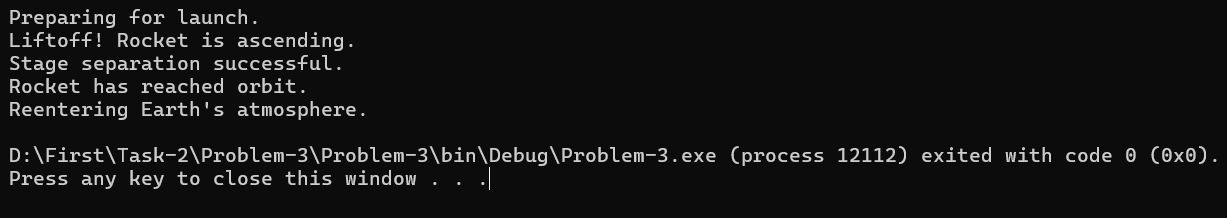
}

}

}

}

**Output:**



**Problem 4:**

Scenario: Neuralink implants operate in multiple modes (Idle, DataProcessing, NeuroStimulation, SleepMode). The implant's behavior depends on the input from neural signals. Problem Statement:

 Create an enum ImplantMode to represent different operating states of a Neuralink implant.

 Implement a class NeuralinkDevice with a method SwitchMode(ImplantMode mode) that allows changing modes based on external triggers.

 Simulate a Neuralink device that cycles through different states based on user input.

**Ans:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Problem\_4

{

enum ImplantMode

{

Idle,

DataProcessing,

NeuroStimulation,

SleepMode

}

class NeuralinkDevice

{

private ImplantMode currentMode;

public NeuralinkDevice()

{

currentMode = ImplantMode.Idle;

}

public void SwitchMode(ImplantMode mode)

{

currentMode = mode;

Console.WriteLine($"\nSwitched to mode: {currentMode}");

switch (currentMode)

{

case ImplantMode.Idle:

Console.WriteLine("System is idle. Monitoring neural signals...");

break;

case ImplantMode.DataProcessing:

Console.WriteLine("Processing neural signals...");

break;

case ImplantMode.NeuroStimulation:

Console.WriteLine("Delivering neurostimulation pulses...");

break;

case ImplantMode.SleepMode:

Console.WriteLine("Entering sleep mode to save power...");

break;

}

}

}

class Program

{

static void Main(string[] args)

{

NeuralinkDevice device = new NeuralinkDevice();

bool running = true;

while (running)

{

Console.WriteLine("\nSelect a mode:");

Console.WriteLine("1. Idle");

Console.WriteLine("2. Data Processing");

Console.WriteLine("3. Neuro Stimulation");

Console.WriteLine("4. Sleep Mode");

Console.WriteLine("5. Exit");

Console.Write("Your choice: ");

string input = Console.ReadLine();

switch (input)

{

case "1":

device.SwitchMode(ImplantMode.Idle);

break;

case "2":

device.SwitchMode(ImplantMode.DataProcessing);

break;

case "3":

device.SwitchMode(ImplantMode.NeuroStimulation);

break;

case "4":

device.SwitchMode(ImplantMode.SleepMode);

break;

case "5":

running = false;

Console.WriteLine("Exiting simulation.");

break;

default:

Console.WriteLine("Invalid input. Please try again.");

break;

}

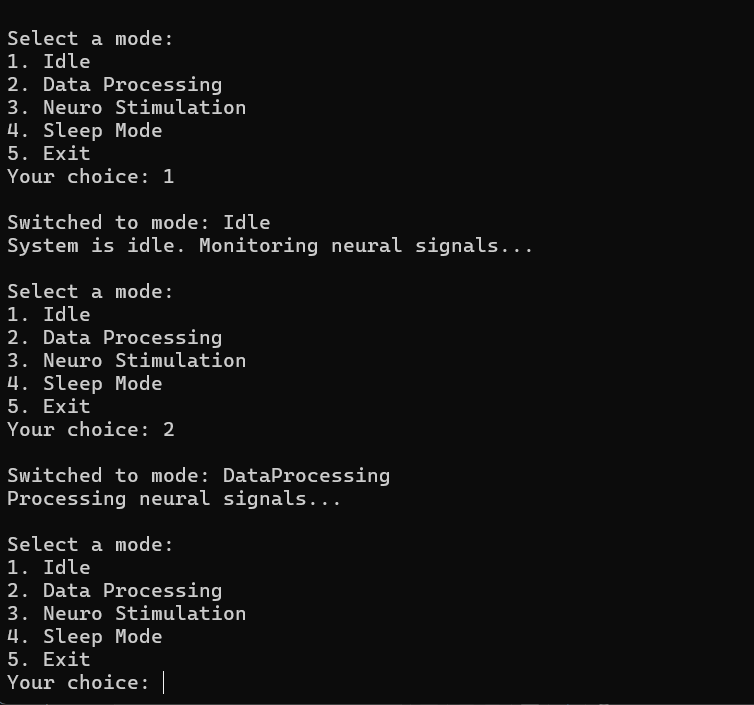
}

}

}

}

**Output:**



**Problem 5:**

Scenario: Tesla’s self-driving system operates at different autonomy levels (Manual, AssistedDriving, Autonomous, EmergencyOverride). Problem Statement:

 Define an enum AutonomyLevel to represent different levels of Tesla’s self-driving system.

 Implement a class TeslaCar with a method SetAutonomy(AutonomyLevel level).

 Implement a driving simulation where the car dynamically changes autonomy levels based on road conditions (e.g., rain, highway, urban).

Ans:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Problem\_5

{

enum AutonomyLevel

{

Manual,

AssistedDriving,

Autonomous,

EmergencyOverride

}

class TeslaCar

{

private AutonomyLevel currentLevel;

public TeslaCar()

{

currentLevel = AutonomyLevel.Manual;

}

public void SetAutonomy(AutonomyLevel level)

{

currentLevel = level;

Console.WriteLine($"\nAutonomy level set to: {currentLevel}");

switch (currentLevel)

{

case AutonomyLevel.Manual:

Console.WriteLine("Driver is in full control.");

break;

case AutonomyLevel.AssistedDriving:

Console.WriteLine("Assistance active: Lane keep and adaptive cruise control.");

break;

case AutonomyLevel.Autonomous:

Console.WriteLine("Tesla is fully driving itself.");

break;

case AutonomyLevel.EmergencyOverride:

Console.WriteLine("Emergency override activated. Taking safety actions!");

break;

}

}

}

class Program

{

static void Main(string[] args)

{

TeslaCar car = new TeslaCar();

bool running = true;

while (running)

{

Console.WriteLine("\nSelect road condition:");

Console.WriteLine("1. Rainy Weather");

Console.WriteLine("2. Highway");

Console.WriteLine("3. Urban City");

Console.WriteLine("4. Emergency Situation");

Console.WriteLine("5. Exit");

Console.Write("Your choice: ");

string input = Console.ReadLine();

switch (input)

{

case "1":

car.SetAutonomy(AutonomyLevel.AssistedDriving);

break;

case "2":

car.SetAutonomy(AutonomyLevel.Autonomous);

break;

case "3":

car.SetAutonomy(AutonomyLevel.Manual);

break;

case "4":

car.SetAutonomy(AutonomyLevel.EmergencyOverride);

break;

case "5":

running = false;

Console.WriteLine("Exiting simulation.");

break;

default:

Console.WriteLine("Invalid input. Please try again.");

break;

}

}

}

}

}

**Output:**

