
Collections + LINQ

Exercise 1: Working with List – Student Names

Problem:

Store and display student names using a `List<string>`.

Instructions:

1. Create a `List<string>` called `students`.
2. Add 5 names to the list.
3. Display all names using a `foreach` loop.
4. Sort the list and display it again.

Exercise 2: Dictionary<TKey, TValue> – Phone Book

Problem:

Build a phone book using `Dictionary<string, string>`.

Instructions:

1. Create a dictionary with **name as key** and **phone number as value**.
2. Add 3 contacts.
3. Display all contacts.
4. Ask the user for a name and display the phone number (if found).

Exercise 3: List of Objects + LINQ – Filter Products

Problem:

Filter products with price > 500 using LINQ.

Instructions:

1. Create a `Product` class with `Name`, `Price`.
2. Create a `List<Product>` and add 5 items.
3. Use LINQ to filter products where `Price > 500`.
4. Display the filtered list.

Exercise 4: LINQ – Get Top 3 Students by Marks

Problem:

Find top 3 scoring students using LINQ.

Instructions:

1. Create a `Student` class with `Name`, `Marks`.
2. Add 6 students to a list.
3. Use LINQ to get the **top 3 by Marks**.
4. Print the result.

Exercise 5: Grouping with LINQ – Group Employees by Department

Problem:

Group employees by department.

Instructions:

1. Create an `Employee` class with `Name`, `Department`.
2. Add 6 employees (across 2–3 departments).
3. Use LINQ `group by` to group them.
4. Print employees under each department.

Exercise 6: LINQ – Count Word Frequency (Strings)

Problem:

Count how many times each word appears in a sentence.

Instructions:

1. Input a sentence like: `"C# is great and C# is fun"`
2. Split it into words.
3. Use LINQ `group by` and `count`.
4. Print:

```
C#: 2
is: 2
great: 1
and: 1
fun: 1
```

Collections + LINQ – 2

Exercise 1: List – Even & Odd Numbers

Problem:

Separate even and odd numbers from a list.

Instructions:

1. Create a `List<int>` with at least 10 numbers.
2. Use LINQ to filter:
   - One list with even numbers
   - One list with odd numbers
3. Print both lists.

Exercise 2: Dictionary – Employee Salary Lookup

Problem:

Build a salary lookup table using `Dictionary<int, decimal>`.

Instructions:

1. Use **Employee ID** as the key, **Salary** as the value.
2. Add 4–5 employees.
3. Ask the user to enter an Employee ID.
4. Display their salary if found.

Exercise 3: LINQ – Count Students Above Average

Problem:

Use LINQ to count how many students scored above the average.

Instructions:

1. Create a `Student` class with `Name`, `Marks`.
2. Add at least 5 students.
3. Calculate average marks using LINQ: `.Average()`
4. Use LINQ to count how many students scored above average.

Exercise 4: LINQ – Find Duplicate Numbers

Problem:

Identify duplicates in a list of numbers.

Instructions:

1. Create a `List<int>` with some repeated numbers.
2. Use LINQ to find numbers that appear more than once.
3. Print the duplicated numbers.

Exercise 5: List of Objects – Sort Products by Price (Descending)

Problem:

Sort products by price using LINQ.

Instructions:

1. Create a `Product` class with `Id`, `Name`, `Price`.
2. Add 5–6 products to a list.
3. Sort the products by price in descending order using LINQ.
4. Print the sorted list.

Exercise 6: LINQ – First Names Starting with a Vowel

Problem:

Filter names that start with a vowel.

Instructions:

1. Create a `List<string>` with 8–10 names.
2. Use LINQ to find names starting with vowels (A, E, I, O, U).
3. Print the result.

Tips:

- Use `Where()`, `OrderByDescending()`, `GroupBy()`, `Count()`, `Average()`
- Prefer `var` when writing LINQ queries for readability
- Keep classes short and to-the-point
