
Exercise 1: Basic Query Expression

Task: Write a LINQ query to select all integers from a list that are greater than 10.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 5, 12, 8, 20, 3 };
        var result = from n in numbers
                     where n > 10
                     select n;
```

```
        Console.WriteLine(string.Join(", ", result));
    }
}
```

Solution: Outputs `12, 20`.

Exercise 2: Lambda Expressions

Task: Use a lambda expression to filter out odd numbers from a list.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
```

```
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
        var evenNumbers = numbers.Where(n => n % 2 == 0);

        Console.WriteLine(string.Join(", ", evenNumbers));
    }
}
```

Solution: Outputs `2, 4`.

Exercise 3: Select Operator

Task: Use the `Select` operator to create a new list where each integer is doubled.

```csharp
using System;
```

```
using System.Collections.Generic;
using System.Linq;

public class Program
{
   public static void Main()
   {
      List<int> numbers = new List<int> { 1, 2, 3, 4 };
      var doubledNumbers = numbers.Select(n => n *
2);

      Console.WriteLine(string.Join(", ",
doubledNumbers));
   }
}
```

Solution: Outputs `2, 4, 6, 8`.

Exercise 4: Sorting with OrderBy

Task: Sort a list of strings by their length using `OrderBy`.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<string> words = new List<string> { "apple", "banana", "cherry", "date" };
        var sortedWords = words.OrderBy(w => w.Length);

        Console.WriteLine(string.Join(", ", sortedWords));
    }
}
```

```

**Solution:** Outputs `date, apple, banana, cherry`.

---

### Exercise 5: GroupBy

**Task:** Group a list of integers by their remainder when divided by 3.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6 };
```
```

```
        var groupedNumbers = numbers.GroupBy(n => n % 3);

        foreach (var group in groupedNumbers)
        {
            Console.WriteLine($"Remainder {group.Key}: {string.Join(", ", group)}");
        }
    }
}
```

Solution: Outputs:

```
Remainder 1: 1, 4, 7
Remainder 2: 2, 5, 8
Remainder 0: 3, 6
```

Exercise 6: IQueryable Interface

Task: Demonstrate the use of `IQueryable` by querying a database context (mocked here).

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
        IQueryable<int> queryableNumbers = numbers.AsQueryable();

        var result = queryableNumbers.Where(n => n > 3);
```

```
        Console.WriteLine(string.Join(", ", result));
    }
}
```

Solution: Outputs `4, 5`.

Exercise 7: PLINQ Basics

Task: Use PLINQ to parallelize a query that filters and orders numbers.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
```

```
    {
        List<int> numbers = new List<int> { 1, 3, 5, 7, 9, 11, 13, 15 };
        var parallelQuery = numbers.AsParallel().Where(n => n % 2 == 0).OrderBy(n => n);

        Console.WriteLine(string.Join(", ", parallelQuery));
    }
}
```

Solution: Outputs `[],` as there are no even numbers.

Exercise 8: FirstOrDefault

Task: Find the first element in a list that is greater than 10. If none is found, return a default value.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 5, 8, 12, 15 };
        var result = numbers.FirstOrDefault(n => n > 10);

        Console.WriteLine(result);
    }
}
```

Solution: Outputs `12`.

Exercise 9: Join Operator

Task: Join two lists of objects based on a common key.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public class Order
```

```
    {
        public int PersonId { get; set; }
        public string Product { get; set; }
    }

    public static void Main()
    {
        List<Person> people = new List<Person>
        {
            new Person { Id = 1, Name = "Alice" },
            new Person { Id = 2, Name = "Bob" }
        };

        List<Order> orders = new List<Order>
        {
            new Order { PersonId = 1, Product = "Laptop"
},
            new Order { PersonId = 2, Product = "Phone" }
        };

        var query = from person in people
```

```
                join order in orders on person.Id equals
order.PersonId
                select new { person.Name, order.Product
};


      foreach (var item in query)
      {
         Console.WriteLine($"{item.Name} bought
{item.Product}");
      }
   }
}
```

Solution: Outputs:

```
Alice bought Laptop
Bob bought Phone
```

Exercise 10: Aggregate Function

Task: Calculate the sum of all integers in a list using the `Aggregate` function.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4 };
        var sum = numbers.Aggregate((total, next) => total + next);

        Console.WriteLine(sum);
    }
}
```

```

**Solution:** Outputs `10`.

---

### Exercise 11: Distinct Elements
**Task:** Remove duplicate elements from a list.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 2, 3, 4,
4 };
        var distinctNumbers = numbers.Distinct();
```
```

```
        Console.WriteLine(string.Join(", ",
distinctNumbers));
    }
}
```

Solution: Outputs `1, 2, 3, 4`.

Exercise 12: Contains

Task: Check if a list contains a specific value.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
```

```
    public static void Main()
    {
        List<string> words = new List<string> { "apple", "banana", "cherry" };
        bool containsBanana = words.Contains("banana");

        Console.WriteLine(containsBanana);
    }
}
```

Solution: Outputs `True`.

Exercise 13: All and Any

Task: Use `All` to check if all elements in a list are positive and `Any` to check if any element is greater than 10.

```csharp
```

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 5, 8, 12 };
        bool allPositive = numbers.All(n => n > 0);
        bool anyGreaterThanTen = numbers.Any(n => n
> 10);

        Console.WriteLine($"All positive: {allPositive}");
        Console.WriteLine($"Any greater than 10:
{anyGreaterThanTen}");
    }
}
```

Solution: Outputs:

```
All positive: True
Any greater than 10: True
```

Exercise 14: SelectMany

Task: Flatten a list of lists into a single list using `SelectMany`.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<List<int>> listOfLists = new List<List<int>>
```

```
        {
            new List<int> { 1, 2 },
            new List<int> { 3, 4 },
            new List<int> { 5, 6 }
        };

        var flattenedList = listOfLists.SelectMany(list => list);

        Console.WriteLine(string.Join(", ", flattenedList));
    }
}
```

Solution: Outputs `1, 2, 3, 4, 5, 6`.

Exercise 15: Take and Skip

Task: Use `Take` to get the first 3 elements and `Skip` to skip the first 3 elements in a list.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6 };
        var firstThree = numbers.Take(3);
        var skipFirstThree = numbers.Skip(3);

        Console.WriteLine("First 3: " + string.Join(", ", firstThree));
        Console.WriteLine("Skip first 3: " + string.Join(", ", skipFirstThree));
    }
```

}
```

**Solution:** Outputs:
```
First 3: 1, 2, 3
Skip first 3: 4, 5, 6
```

---

### Exercise 16: ToList and ToDictionary

**Task:** Convert a list of objects to a dictionary using `ToDictionary`.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program

```
{
    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public static void Main()
    {
        List<Person> people = new List<Person>
        {
            new Person { Id = 1, Name = "Alice" },
            new Person { Id = 2, Name = "Bob" }
        };

        var peopleDictionary = people.ToDictionary(p
=> p.Id, p => p.Name);

        foreach (var kvp in peopleDictionary)
        {
```

```
            Console.WriteLine($"Id: {kvp.Key}, Name: {kvp.Value}");
        }
    }
}
```

Solution: Outputs:

```
Id: 1, Name: Alice
Id: 2, Name: Bob
```

Exercise 17: Using DefaultIfEmpty

Task: Use `DefaultIfEmpty` to handle empty sequences.

```csharp
using System;
```

```
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int>();
        var result = numbers.DefaultIfEmpty(0);

        Console.WriteLine(string.Join(", ", result));
    }
}
```

Solution: Outputs `0`.

Exercise 18: ElementAtOrDefault

Task: Use `ElementAtOrDefault` to access an element at a specific index.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 10, 20, 30 };
        var element = numbers.ElementAtOrDefault(2);

        Console.WriteLine(element);
    }
}
```

Solution: Outputs `30`.

Exercise 19: Using TakeWhile and SkipWhile

Task: Use `TakeWhile` to take elements as long as a condition is met, and `SkipWhile` to skip elements as long as a condition is met.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6 };
        var takeWhile = numbers.TakeWhile(n => n < 4);
        var skipWhile = numbers.SkipWhile(n => n < 4);
```

```
        Console.WriteLine("Take while: " + string.Join(", ", takeWhile));
        Console.WriteLine("Skip while: " + string.Join(", ", skipWhile));
    }
}
```

Solution: Outputs:

```
Take while: 1, 2, 3
Skip while: 4, 5, 6
```

Exercise 20: Aggregate with Seed

Task: Use `Aggregate` with a seed value to calculate the product of all integers in a list.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 2, 3, 4 };
        var product = numbers.Aggregate(1, (total, next) => total * next);

        Console.WriteLine(product);
    }
}
```

Solution: Outputs `24`.
