[1310] LINQ

LINQ stands for "Language Integrated Query."

LINQ can be used to parse, summarize, and query data stored any C# collection that implements IEnumerable (which is pretty much all of them). LINQ can also be used to access external data sources like SQL databases.

While LINQ has some similarities to SQL, the two are actually different when you get into the details. That said, a basic understanding of SQL queries can help you understand LINQ queries too.

Simple Example:

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

class LinqDemo
{
    static void Main()
    {
        string[] companies = { "Apple", "Intel", "Microsoft", "Amazon", "Google", "Facebook" };

        IEnumerable<string> ooCompanies = companies.Where(n => n.Contains("oo"));
        foreach (string company in ooCompanies)
            Console.WriteLine(company);
    }
}

Output:
Google
Facebook
```

Note: Often, C# programmers will use "var" instead of "IEnumerable<>" when working with LINQ. The book uses "IEnumerable<>".

In Class Exercise

Change the program above so that it displays a list of all the companies with a name that is more than 5 characters long.

Fluent Syntax

Fluent syntax allows you to chain several different method calls together with dots. LINQ can use Fluent Syntax to create more complex queries. Here's an example:

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

```
class LinqDemo
{
    static void Main()
    {
        string[] companies = { "Apple", "Intel", "Microsoft", "Amazon", "Google", "Facebook" };

        IEnumerable<string> selectedCompanies = companies
                .Where(c => c.Contains("o"))
                .OrderByDescending(c => c)
                .Select(c => c.ToUpper());
        foreach (string company in selectedCompanies)
            Console.WriteLine(company);
    }
}
```

Output:

```
MICROSOFT
GOOGLE
FACEBOOK
AMAZON
```

Note: OrderByDescending() sorts in reverse order.

In-Class Exercise

Change the program above so that it displays a sorted, lower-case list of all the companies whose name ends with an "e".

Query Syntax

C# also provides a way to write LINQ queries so that they look VERY similar to SQL queries. This syntax pretty much breaks ALL the rules for regular C# syntax so, when you see it, just "go with it" and ignore the fact that it looks really strange. Here's the previous program using Query Syntax:

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

class LinqDemo
{
    static void Main()
    {
        string[] companies = { "Apple", "Intel", "Microsoft", "Amazon", "Google", "Facebook" };

        IEnumerable<string> selectedCompanies =
                from c in companies
                where c.Contains("o")
                orderby c descending
                select c.ToUpper();
        foreach (string company in selectedCompanies)
            Console.WriteLine(company);
    }
```

```
}

Output:
MICROSOFT
GOOGLE
FACEBOOK
AMAZON
```

Note: Unlike SQL, the select part of the query MUST be at the end instead of the beginning.
Also, note that orderby has "descending" after the field in Query Syntax (just like SQL).

In-Class Exercise

Convert the previous program you wrote so that it uses Query Syntax instead of Fluid Syntax.

Select Projections

Projections are what the Select() method does. They are the "results" of the query including the format and the type. So far, our projections have all be simple types (string). LINQ has the ability to project into more complex types AND the ability to create new types as needed.

Select (combined with "new") can put the results of a LINQ query into objects of an existing class OR Select can create an "Annoymous Type" and stick the results in there. Here's an example:

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

namespace LWTech.Testing
{
   class LinqDemo
   {
      static void Main()
      {
         string[] companies = { "Apple", "Intel", "Microsoft", "Amazon", "Google", "Facebook" };

         var selectedCompanies =
            from c in companies
            orderby c descending
            select new
            {
               OriginalName = c,
               UpperCaseName = c.ToUpper(),
               LowerCaseName = c.ToLower(),
               NameLength = c.Length
            };
         foreach (var company in selectedCompanies)
            Console.WriteLine($"{company.UpperCaseName} ({company.NameLength}) => {company.LowerCaseName}");
      }

   }
}
```

```
Output:
MICROSOFT (9) => microsoft
INTEL (5) => intel
GOOGLE (6) => google
FACEBOOK (8) => facebook
APPLE (5) => apple
AMAZON (6) => amazon
```

Note: You MUST use "var" in this case for the type since the actual type is "annoymous" and only known by the compiler.