LINQ and Files

9.1 Introduction to LINQ

- **LINQ** (**Language-Integrated Query**) capabilities

  Allow you to write **query expressions** that retrieve information from a *variety* of data sources, not just databases.

- **LINQ to Objects** can be used to **filter** arrays and `List`s

  Selecting elements that satisfy a set of conditions

- **LINQ** is similar to SQL
  - An international language used to perform **queries** and to manipulate data.
  - Queries are like to request information that satisfies given criteria.

9.1 Introduction to LINQ

- The syntax of LINQ in built into C#, but LINQ queries may be used in many different contexts.
- A **LINQ provider** is a set of classes that implement LINQ operations and enable programs to interact with *data sources* to perform tasks such as *projecting*, *sorting*, *grouping* and *filtering* elements.

- The `System.Linq` namespace contains the LINQ to Objects provider.

- *Example*: 24-1-LINQWithSimpleTypeArray

9.2 Querying an Array Using LINQ

- LINQ specifies the conditions that selected elements must satisfy.
  - This is known as ***declarative programming***
    - It does not specify how a task are performed.
    - The C# complier generates all the necessary code.
  - As opposed to ***imperative programming***
    - In which you specify the actual steps to perform a task (which we've been doing so far).

9.2 Querying an Array Using LINQ

- A LINQ query begins with a `from` **clause**, which specifies a **range variable** and the data source to query.

- **Implicitly typed local variables**
  - Enables the compiler to *infer* a local variable's type based on the context in which it's used.
  - i.e. `from` statement and `var` keyword
  - You can declare a local variable and let the compiler *infer* the variable's type based on the variable's initializer.
  -
```
var x = from value in values
        where value > 4
        select value;
```
  -
```
var x = 1;
```

9.2 Querying an Array Using LINQ

- `where` clause
  - predicate
    - The condition in front of the where clause.
    - It is an expression that takes an element of a collection and returns `true` or `false` by testing a condition on that element.
    - The conditional AND (&&) operator can be used to combine conditions.

- `select` clause
  - Determines what value appears in the results.
  - Can be used to select a member of the range variable as well.

9.2 Querying an Array Using LINQ

- `orderby` clause

  Sorts the query results in *ascending* order.
  - `descending` modifier sorts the results in *descending* order.
  - Can sort the results according to multiple properties, specified in a comma-separated list.

- Any value that can be compared with other values of the same type may be used with the `orderby` clause.

9.3 Query Result's Methods

- `Any` method returns
  - `true` if there is at least one element, and
  - `false` if there are no elements.

- `First` method returns
  - The first element in the result.

- `Count` method returns
  - The number of elements in the results.

- `Distinct` method
  - Removes duplicate elements.

9.3 Querying an Array of Employee Objects Using LINQ

- The select clause can create a new object of **anonymous type** (a type with no name).
  - The compiler generates it.

```
var names =
 from e in employees
 select new { e.FirstName, Last = e.LastName };
```

- By default, the name of the property being selected is used as the property's name in the result.
- You can specify a different name for the property inside the anonymous type definition.

9.3 Querying an Array of Employee Objects Using LINQ

- Implicitly typed local variables allow you to use anonymous types because you do not have to explicitly state the type when declaring such variables.

- When the compiler creates an anonymous type, it automatically generates a `ToString` method that returns a `string` representation of the object.

9.3 Querying an Array of Employee Objects Using LINQ

- **Projection**

  Performs a transformation on the data.

  Creates new objects containing only the `FirstName` and `Last` properties.

- Transformations can also manipulate the data.
  - For example, you could give all employees a 10% raise by multiplying their `MonthlySalary` properties by `1.1`.

- *Practice*: 24-2-LINQWithArrayOfObjects-Practice
  - Download and follow instructions

9.4 Collection class `List<T>`

- `List<T>` is called a **generic class** because it can be used with any type of object.
  - Namespace `System.Collections.Generic`

- `List` vs `Array`
  - `List` does not need to be reallocated to change its size.

Class `List<T>`

Method or property	Description
`Add`	Adds an element to the end of the `List`.
`Capacity`	Property that gets or sets the number of elements a `List` can store without resizing.
`Clear`	Removes all the elements from the `List`.
`Contains`	Returns `true` if the `List` contains the specified element and `false` otherwise.
`Count`	Property that returns the number of elements stored in the `List`.
`IndexOf`	Returns the index of the first occurrence of the specified value in the `List`.
`Insert`	Inserts an element at the specified index.
`Remove`	Removes the first occurrence of the specified value.
`RemoveAt`	Removes the element at the specified index.
`RemoveRange`	Removes a specified number of elements starting at a specified index.
`Sort`	Sorts the `List`.
`TrimExcess`	Sets the `Capacity` of the `List` to the number of elements the `List` currently contains (`Count`).

9.5 Querying a Generic Collection Using LINQ

- You can use LINQ to Objects to query `List`s just as arrays.

- LINQ's `let` clause
  - Can be used to create a new range variable to store a temporary result for use later in the LINQ query.

- *Example*: 24-3-LINQWithListCollection

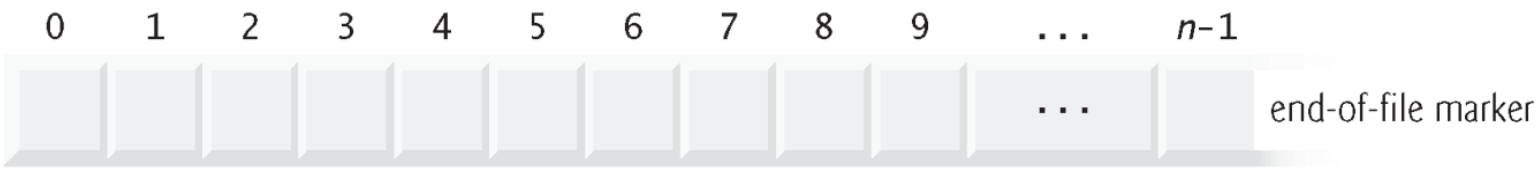
9.5 Querying a Generic Collection Using LINQ

- LINQ uses **deferred execution**
  - The query executes *only* when you access the results (i.e. iterating over them or using `count`)
  - *Not* when you define the query.

- LINQ extension methods `ToArray` and `ToList`
  - Immediately execute the query on which they are called.
  - Return an array or `List<T>` respectively.
  - These methods execute the query only once, improving efficiency.

17.3 Files and Streams

- C# views each file as a sequential stream of bytes.



- Each file ends either
  - with an end-of-file marker or
  - at a specific byte number that's recorded in a system-maintained administrative data structure.

17.3 Files and Streams

- When a file is opened
  - an object is created and
  - a stream is associated with the object.

- When a console app executes, the runtime environment creates
  - Three stream objects that are accessible via properties `Console.Out`, `Console.In`, `Console.Error`
  - In order to facilitate communication between a program and a particular file or device.

17.3 Files and Streams

- `Console.In`
  - standard input stream object
  - Enables a program to input data from the keyboard.
- `Console.Out`
  - standard output stream object
  - Enables a program to output data to the screen.
- `Console.Error`
  - standard error stream object
  - Enables a program to output error messages to the screen.

17.3 Files and Streams

- `System.IO` namespace includes
  - `StreamReader` (for text input from a file)
    - For text input from a file
      - `ReadLine();` `ReadToEnd();`
  - `StreamWriter` (for text output to a file)
    - For text output to a file
      - `WriteLine(string);` `Write(string);`
  - `FileStream` (for both input from and output to a file)

17.4 Classes File and Directory

- Classes `File` and `Directory`

  Enable programs to manipulate files and directories on disk.

- Class `File`

  Can determine information about files.

  Can be used to open files for reading or writing.

17.4 Directory Class

static Method	Description
CreateDirectory	Creates a directory and returns its associated DirectoryInfo object.
Delete	Deletes the specified directory.
Exists	Returns true if the specified directory exists and false otherwise.
GetDirectories	Returns a string array containing the names of the subdirectories in the specified directory.
GetFiles	Returns a string array containing the names of the files in the specified directory.
GetCreationTime	Returns a DateTime object representing when the directory was created.
GetLastAccessTime	Returns a DateTime object representing when the directory was last accessed.
GetLastWriteTime	Returns a DateTime object representing when items were last written to the directory.
Move	Moves the specified directory to a specified location.

17.4 File Class

static Method	Description
AppendText	Returns a StreamWriter that appends text to an existing file or creates a file if one does not exist.
Copy	Copies a file to a new file.
Create	Creates a file and returns its associated FileStream.
CreateText	Creates a text file and returns its associated StreamWriter.
Delete	Deletes the specified file.
Exists	Returns true if the specified file exists and false otherwise.
GetCreationTime	Returns a DateTime object representing when the file was created.
GetLastAccessTime	Returns a DateTime object representing when the file was last accessed.
GetLastWriteTime	Returns a DateTime object representing when the file was last modified.
Move	Moves the specified file to a specified location.
Open	Returns a FileStream associated with the specified file and equipped with the specified read/write permissions.
OpenRead	Returns a read-only FileStream associated with the specified file.
OpenText	Returns a StreamReader associated with the specified file.
OpenWrite	Returns a write FileStream associated with the specified file.

17.4 DirectoryInfo Class

- To create an object of `DirectoryInfo`:

```
myDirectoryInfoObject = new DirectoryInfo(path);
```

Or

```
myDirectoryInfoObject =
Directory.CreateDirectory(path);
```

- Methods:
  - `myDirectoryInfoObject.GetDirectories();`
    - Returns an array of `DirectoryInfo` containing all subdirectories.
  - `myDirectoryInfoObject.GetFiles();`
    - Returns an array of `FileInfo` containing all files.
- Properties:
  - `myDirectoryInfoObject.Name`
  - `myDirectoryInfoObject.FullName`
  - `myFileInfoObject.Name`
  - `myDirectoryInfoObject.Parent`

17.4 `Path` and `FileInfo` Classes

- `Path.Combine(path, subdirectory)`
  - Combines two strings into a path (path/subdirectory).
- `Path.GetExtension( file )`
  - Return the extension of the file
- Class `FileInfo` has pretty similar constructors and methods to `DirectoryInfo` class.

17.4 `Dictionary` Collection

- It is a collection of *key-value pairs*, in which each *key* has a corresponding *value*.
- Include namespace `System.Collections.Generic`
  - It is a generic class.
- Property `Key`
  - Get a collection of all keys
- Use indexers ([*key*]) to get the corresponding *value*.
- Method `Clear`
  - Clear the dictionary
- Method `ContainsKey`
  - Returns true if a specified key exists in the dictionary
- Method `Add`
  - To add a new key-value pair