Lab 3 – File I/O and Text Manipulation

***data.csv* can be found on D2L under the Assignment submodule in Week 5.**
Submit the link to your GitHub repository containing the solution to D2L by the specified due date.

Part 1: Create a Class Called FileRoot

As we know, when we run a program, the program runs from the bin/debug/net6.0 folder. If I had a file stored at the root of the project and if the program is running, it wouldn't know where it is unless it came out of that folder. Before implementing the classes below, create a class that will help you get from the bin folder to the project's root. *Hint: Directory class and its static methods would help here. See in-class example.*

Part 2: Create a project to convert CSV to PSV

Using the provided CSV file (data.csv), read in the data, create the related objects based on the following descriptions, and convert the data to pipe-separated values (psv).

1. Create the classes shown below.
2. Load the data from the CSV and create the **Person** object from it.
3. Add each **Person** object to a **List<Person>** called **people**
4. The **Person** and **Address** class' **ToString()** methods should **represent the data using the pipe (|) separator** between each of the properties.
5. Implement the **IComparable** interface in the **Person** class (sort by **LastName**)
6. In the driver, call the **Sort()** method on the collection of Person objects created in step 3
    a. This should sort on LastName if the IComparable method was implemented correctly
7. Create a new file called *data.psv* that contains each element in the people collection
    a. Use **StreamWriter**, and write each element in the List using **WriteLine()**
    b. *data.psv* should display the same data as *data.csv*, but will contain pipe characters (|) rather than commas, and will be sorted based on each **Person** object's **LastName**

Please note, I do not care what you want to do regarding methods, constructors, or whatever, but the described properties must be created, and the **ToString()** methods for these classes need the functionality described in step 4 above.

Person class

- Properties
    - FirstName: string -> public string FirstName {get; init;}
    - LastName: string
    - Address: Address

Address Class

- Properties
    - StreetAddress: string
    - City: string
    - State: string
    - PostalCode: string