# Purpose

The purpose of this assignment is to get experience with the strings, regular expressions, Lists, operators, and indexers.

# Specifications

## The Name Class

A person’s name may be in any form typically used for names in this country. Examples include the following.

Will E. Makit I. M. Smart chauncey c. chauncey, iii

Betty Wont Kute, U. R. Morton Downey, Jr

Ima Raven Nutt, IV Colder, I. Ben, Jr. McPherson Mack Pherson, MD

Thayer B. Fuddled Badly, Claude Sue N. Lawyer, JD

U. Ara Nice Pursun Dr. Heza Dummox I. N. Stein, PhD

SAM SMITH-JONES mckensie j. quincy macdonald o’reilly, riley

Develop a ***Name*** class that keeps track of the name and its components. It should provide the functionality to decompose and recompose/format any name in any of the various forms. Only “American” types of names need be handled.

The ***Name*** class should have a ***default*** ***constructor***, a ***copy*** ***constructor***, and a ***parameterized*** ***constructor***. The third of these constructors should accept a ***string*** representing a ***name*** in any of the permissible formats and decompose it appropriately into its components. The case of the letters in the name components should be adjusted so that, when displayed, the names appear correctly. The **Tools.Tokenize** method may be of use.

The ***Name*** class should have methods that return strings. One should contain the name in the form of first name first (such as I. Ben Colder, Jr.). Another should return the name in the form of last name followed by a comma, followed by the rest of the name (such as Makit, Will E., PhD).

The ***Name*** class must implement the ***IEquatable<Name>*** and ***IComparable<Name>*** interfaces according to the standards and techniques discussed in class. The comparisons of two ***Name*** objects should be based on a ***String*** comparison of a combination of the ***Last***, ***Rest***, and ***Suffix*** properties, in that order. The comparisons should ignore case differences.

## NameList Class

* Create a ***NameList*** class that represents an ***indexable*** ***List*** of ***Name*** objects.
* The ***NameList*** class should maintain the collection internally through the use of a private ***List<Name>*** instance.
* Because the private internal ***List*** is not accessible to a user of the ***NameList*** class, you must provide one or more properties for the ***NameList*** class that allow access to any important properties of the ***List<Name>*** object such as ***Count***.
* Define the ***+ operator*** for the ***NameList*** class to allow one to add a ***Name*** to the internal ***List*** – if it is not already there.
* Define the ***- operator*** for the ***NameList*** class to allow one to remove a ***Name*** from the internal ***List*** – if it is present. You will need to decide how this operator should behave if the ***Name*** is not found in the internal ***List***. Be prepared to explain the rationale for your decision.
* Define one or more ***Indexers*** on the ***NameList*** class that will allow the user to retrieve/replace a single ***Name*** in the internal ***List***.
* There should be two methods that can return sorted ***List<String>***. One should return a sorted ***List<String>*** containing the all of the names, last name first. The other should return a sorted ***List<String>*** containing all of the names, first name first.
* Use the methods of the ***List<T>*** class and the ***Name*** class where you can to avoid brute force approaches.

## Driver Class

* The driver should be menu-driven. You may use the ***Menu*** class I posted on the course site, or you may use your own.
* The driver should allow the user to select a text file containing any number of names, one per line, in any of the formats allowed above. An ***OpenFileDialog*** should be used for this purpose.
* The driver should be able to input the names from the file specified above and build a ***NameList*** from them.
* The driver should allow the user to display a single name either ***last name first***, ***first name first***, or as ***originally*** ***read*** from the file.
* The driver should allow the user to request a display of a sorted list of all of the ***Names*** in the ***NameList*** either by last name first or by first name first.
* The program should allow the user to ***add*** a ***new*** ***name*** to the internal list.
* The program should allow the user to ***delete*** a ***name*** from the internal list.
* When the program starts, the driver should display a ***welcome*** ***message*** using an appropriate ***static*** method in your ***Tools*** class. It should then prompt the user to input his/her ***credentials*** including ***name***, ***email*** ***address***, and ***phone*** ***number***. Create a ***Name*** object for the name. Use ***regular*** ***expressions*** to validate the correct format is used for the ***email*** ***address*** and ***phone*** ***number***, rejecting any that are in an incorrect format.
* When the program is ready to terminate, it should display a personalized ***Goodbye*** message using an appropriate ***static*** method from your ***Tools*** class. The message should include the ***user’s*** ***credentials*** entered at the beginning of the program. They should be formatted in such a way that they blend into the message comfortably.
* Before the ***driver*** ends, it should allow the use to ***save*** the current ***NameList*** (the ***Original*** ***Names***) with any changes that have been made into a ***text*** ***file***. The user should be able to select the text file into which the file is written using a ***SaveFileDialog***.

## Tools Class

* Continue to add ***static*** methods to your ***Tools*** class as you find needs for methods that may be useful in the future.
* Following are two instructor-provided utility methods you may want to add to your ***Tools*** class.

///<summary>  
 ///*Display a Press Any Key to ... message at the bottom of the screen*  
 ///</summary>  
 ///<param name="strVerb">*term in the Press Any Key to ... message; default: "continue . . ."*</param>  
 public static void PressAnyKey (string strVerb **=** **"continue ..."**)  
 {  
 Console**.**ForegroundColor **=** ConsoleColor**.**Red;  
 if (Console**.**CursorTop **<** Console**.**WindowHeight **-** 1)  
 Console**.**SetCursorPosition (0, Console**.**WindowHeight **-** 1);  
 else  
 Console**.**SetCursorPosition (0, Console**.**CursorTop **+** 2);  
   
 Console**.**Write (**"Press any key to "** **+** strVerb);  
 Console**.**ReadKey ( );  
 Console**.**Clear ( );  
 Console**.**ForegroundColor **=** ConsoleColor**.**Blue;  
 } *// End PressAnyKey*

///<summary>  
 ///*Skip n lines in the console window*  
 ///</summary>  
 ///<param name="n">*the number of lines to skip - defaults to 1*</param>  
 public static void Skip (int n **=** 1)  
 {  
 for (int i **=** 0; i **<** n; i**++**)  
 {  
 Console**.**WriteLine ( );  
 }  
 }

# Deliverables

Follow the instructions found on the Fact Sheet for the course that was distributed at the first class meeting and that is posted on the course web site. Turn in the entire project zipped, less the ***bin*** and ***obj folders*** that you must delete. Please include your text file containing the names with which you tested your project.