ITE 365 Lab09: Associative Collections

Adam Lewis

June 11, 2018

Contents

1 Objectives

1. Gain an understanding of the `System.Collections` frameworks, particularly the `SortedDictionary` collection

2 Background

Students in CS372 experience the unmitigated joy of having to implement your own set of classes to work with the data structures discussed in that course. Now that you've done that once, you can start to take advantage of the fact that language and platform providers provide implementations of these classes.

2.1 `System.Collection`

The .NET Framework Class Library provides a set of "Collection" classes that implement many of the common data structures required by applications. As this is provided as part of .NET, these classes are available to any language that works with the .NET Common Language Runtime (Visual C++, Visual BASIC, and, of course, C#).

The Collection classes are built on top of C# *generics* feature; that means that you create a collection of some generic type that you parameterize at run-time.

2.2 Associative Data Structures

We are quite used to working with data structures such as Arrays or Lists that can be accessed as if they were stored in a contiguous block of memory. We can refer to each element by an index number.

For some cases, we need to be able to access data in a data structure by keys other than numeric keys. For example, think about one might look up a word in a dictionary. The items in the dictionary are indexed by word rather than number. This is an example of an associative data structure.

The .NET Framework Class Library provides a number of classes that implement different forms of associative data structures. For example, we have a number of `Dictionary` classes that store data in key-value pairs (just like a printed dictionary). Let's take a look at how we can use the `SortedDictionary` class to keep track of the number of occurrences of each word in a string.

3 Instructions

We will be using the .NET Framework Class Library's support for Regular Expressions to help us split the text up into words. Just roll with that part of the lab.

3.1 Setup

Start Visual Studio and create a new Console application named "SortedDictionaryTest". In your main class file, make certain to include the `System.Collections.Generic` and `System.Text.RegularExpressions` frameworks.

3.2 Things for the `main()`

Add the following code to the `Main()` function:

```
// create sorted dictionary based on user input
SortedDictionary< string, int > dictionary = CollectWords();

// display sorted dictionary content
DisplayDictionary( dictionary );
```

3.3 The heavy lifting: `CollectWords`

Add the following to your file:

```
// create sorted dictionary from user input
private static SortedDictionary< string, int > CollectWords()
{
   // create a new sorted dictionary
   SortedDictionary< string, int > dictionary =
      new SortedDictionary< string, int >();

   Console.WriteLine( "Enter a string: " ); // prompt for user input
   string input = Console.ReadLine(); // get input

   // The processing work will go here....

   return dictionary;
} // end method CollectWords
```

Now we need to add the code to fill the dictionary:

```
// split input text into tokens
string[] words = Regex.Split( input, @"\s+" );

// processing input words
foreach ( var word in words )
{
   string wordKey = word.ToLower(); // get word in lowercase

   // if the dictionary contains the word
   if ( dictionary.ContainsKey( wordKey ) )
   {
```

```
      ++dictionary[ wordKey ];
   } // end if
   else
      // add new word with a count of 1 to the dictionary
      dictionary.Add( wordKey, 1 );
} // end foreach
```

3.4 Printing generically

Now let's take advantage of C# generics to let us write a function that will print ANY `Dictionary` object!

```
// display dictionary content
private static void DisplayDictionary< K, V >(
   SortedDictionary< K, V > dictionary )
{
   Console.WriteLine( "\nSorted dictionary contains:\n{0,-12}{1,-12}",
      "Key:", "Value:" );

   // generate output for each key in the sorted dictionary
   // by iterating through the Keys property with a foreach statement
   foreach ( K key in dictionary.Keys )
      Console.WriteLine( "{0,-12}{1,-12}", key, dictionary[ key ] );

   Console.WriteLine( "\nsize: {0}", dictionary.Count );
} // end method DisplayDictionary
```

4 Submission instructions

Combine a copy of your source code and a screen-shot of your program into a PDF file (you can use Microsoft Word to do this) and attach it to the assignment on Blackboard.