WORKSHEET 7: TRAVERSAL

Version 1.0
Computing
COMP110

Dr Ed Powley

Introduction

In this worksheet, you will implement a number of C# functions to analyse a directory of files.

A file system on a computer is organised into *directories* (or *folders*) and *files*. A directory can contain files as well as recursively containing other directories. We refer to the files *below* a directory as the files contained within that directory itself, within all subdirectories, within all subdirectories of subdirectories, and so on.

To complete this worksheet:

(a) **Fork** the skeleton project and **open** the project in Visual Studio.
(b) **Implement** the following methods of the `DirectoryUtils` class:
  (i) `GetTotalSize()`, which should return the total size (in bytes) of all the files below the given directory.
  (ii) `CountFiles()`, which should return the total number of files below the given directory. This should only count files, not directories.
  (iii) `GetDepth()`, which should return the nested "depth" of the directory: this is the maximum number of subdirectory levels between the top-level directory and a file. For example: a directory containing only files has a depth of 0; a directory whose subdirectories have depth 0 has depth 1; a directory whose subdirectories have depth at most 1 has depth 2.
  (iv) `GetSmallestFile()` and `GetLargestFile()`, which should return the smallest and largest file below the given directory respectively. These functions return a `Tuple<string, long>`, whose first element is the path to the file and whose second element is the size of the file in bytes. If there is more than one file with equal smallest or largest size, the function will return one of them arbitrarily.
  (v) `GetFilesOfSize()`, which should get all files whose size is equal to the given size in bytes. This function returns an `IEnumerable<string>`, a sequence of file paths. For example it may return a `List<string>` or a `string[]`, or it may be implemented as an iterator using `yield return`.
(c) **Test** your implementation. A small directory structure (based on an asset pack from Kenney[1]) is provided in the base repository for you to test on, and a set of unit tests is provided based on this. You are also encouraged to test your implementations on other directory structures.

The skeleton project contains a file `Program.cs`, which you may find this useful when testing your code. There is also a unit test project which defines a series of tests using the NUnit[2] framework; these can be run locally within Visual Studio, and will also be run automatically via TravisCI when you submit a pull request on GitHub.

The provided `DirectoryUtils` class has some helper methods which you may find useful:

[1] `https://www.kenney.nl/assets/pixel-vehicle-pack`
[2] `https://nunit.org`

- `GetFileSize()` returns the size of the given file in bytes.
- `IsDirectory()` returns true if the given path refers to a directory — for example, this can be used to distinguish between directories and files.

The following methods from the .NET standard library, all in the `System.IO` namespace, may also be useful:

- `Directory.GetDirectories()` returns an array containing the directories within the given directory.
- `Directory.GetFiles()` returns an array containing the files within the given directory.
- `Directory.GetFileSystemEntries()` returns an array containing both the files and the directories within the given directory.

Submission instructions

Begin by **forking** the GitHub repository at the following URL:

`https://github.com/Falmouth-Games-Academy/comp110-worksheet-7`

Edit `DirectoryUtils.cs`, implementing the required functions. When you have finished, open a **pull request**.

Do not move or rename `DirectoryUtils.cs`**, and do not edit or delete any of the files in the repository except for** `DirectoryUtils.cs` **and** `Program.cs`. Doing so will interfere with the automated testing scripts used to check your submission for correctness, and as a result may lead to you losing marks.

Upload all material to GitHub and open a pull request by the deadline listed on LearningSpace.

Marking criteria

Remember that **it is better to submit incomplete work than to submit nothing at all**.

Your work will be marked according to the following criteria:

- **Functional coherence**. Is your implementation correct? Your code will be run through TravisCI to verify that it gives the correct results for a sample of input values.
- **Sophistication**. Have you made use of appropriate code structures and data structures? Note the emphasis is on **appropriate**; extra credit will **not** be given for unnecessarily complex solutions. However extra credit **will** be given for solutions that allow for efficient code reuse between functions, for example using iterator functions and/or LINQ.
- **Maintainability: readability**. Is your code well commented? Are your identifier names appropriate and descriptive? Have you adhered to appropriate coding standards?