COLORADO SCHOOL OF MINES
ELECTRICAL ENGINEERING & COMPUTER SCIENCE DEPARTMENT

CSCI-410
Elements of Computing Systems
Spring 2014

PY-02 (rev. 1)

In this assignment, we will step aside from the Header tools for a moment and focus on working with a computer's file system in a way that is (hopefully) platform independent so that your Python scripts can run on Windows or *nix. We will also tackle processing command line arguments and performing file I/O.

We will limit ourselves to a pretty narrow scope that matches the scope needed for the ECS programming assignments later in the course. As such, we may not always do things in the most efficient or elegant way.

The goal of this project, which is admittedly quite contrived, is quite straightforward. We want to have our program run from the command line as follows:

```
prompt> py02 source level
```

Where **`prompt`** is the command prompt, **`source`** is either a file name (with extension) or a directory name, and **`level`** is an optional flag that indicates what information will be in the output report.

For simplicity, we will require that directories not contain any periods and that filenames only contain one. This is not a necessary restriction, merely one that we choose to impose to make our lives easier.

To focus us a little better, we will use the Project 12 directory as our test directory. The best way to do this is to copy the entire directory into your PY02 directory. If you examine this directory you will see that it contains several .jack files as well as several directories. What we will do if write some Python scripts that will let us process these files either individually or as a group.

You will not do very much with the files – the objective of this assignment is pretty well met if you are simply able to locate and access the files. So, depending on the command line options supplied by the user, you will either simply list the filenames that are found or list the filenames along with the number of lines and/or characters in the file.

FileSet class specification

Routine	Arguments	Returns	Description
constuctor	filename (str), ext (str)	--	Builds a set/list of all the files that match the filename/extension parameters.
baseName	--	basename	Returns the basename.
hasMoreFiles	--	Boolean	True if more files yet to be processed
nextFile	--	filespec (str)	Returns name of next file and removes it from the set/list of unprocessed files.
report	--	--	Outputs a summary to the screen.

The table above uses the same format as the class specifications given in the ECS projects, so this should help you get familiar with them. Furthermore, you will be able to use this class in your ECS projects to deal with handling your input files so that you can deal with processing them. Note that these functions are the public functions – you may add private functions at will.

Note that the **`report`** method in the FileSet class is NOT the method you will use to generate the report asked for in this assignment. This class serves a narrowly defined purpose, namely it:

- Determines if the user supplied a specific filename or the name of a directory.
- Builds and maintains the set of files that the user wishes to process.
- Furnishes the file specifications to the user one at a time.
- Provides a means for the user to tell if there are more files yet to process.
- Allows the user to generate a simple report of the files in the set.

You will use a second class, FileScanner, (described later) to process the files and prepare the final report.

Except for **`constructor`**, your function/method names are expected to use the case as given in the specification. The constructor in Python must be named "**`__init__`**()", with two underscore characters before and after the word "**`init`**". Unlike some object oriented languages, including Jack, the Python constructor takes **`self`** as its first argument even though it is not called by dereferencing an instantiated object -- which it can't be since its purpose is to construct the object! How Python does this is that when the class name itself is called as if it were a function, for instance

```
myfileset = FileSet(filename, "jack")
```

Python first allocates memory for the data structure associated with an object of that class and then passes a pointer to that structure, along with the rest of the explicit arguments supplied, to the class' constructor.

Here are more detailed requirements for each public method in the FileSet class.

`constructor` The first argument is either a filename (e.g., **`"apple.jack"`**), or a directory name (e.g., **`"apple"`**). The second argument is an extension (e.g., **`"jack"`**). If the first argument is a filename whose extension matches the second argument, then the file set will consist of a single file in the current working directory. If the first argument is a directory, then the file set will consist of all files in that directory (which is itself in the current working directory) having an extension that matches the second argument. If no matching files exist (or if the named directory does not exist), then the file set will be empty.

`baseName` This method returns the filename (w/o extension) or the directory name in use. For the above example, it would return **`"apple"`** in either case.

`hasMoreFiles` The object keeps track of which files in the set have not yet been processed and returns True if there are still unprocessed files and False if there are not.

`nextFile` This function returns the name of the next unprocessed file. Once this function has been called, the object assumes that that file can now be considered to have been processed. This function should not be called unless a prior call to **`hasMoreFiles()`** indicates that there are still unprocessed files.

`report` This function generates a very simple report that gives the base name, whether the file set is for a single file or a directory, how many files are in the set, and what they are. Examples are below.

```
Processing DIRECTORY
Base: 12
Type: jack
Files: 8
  Array.jack
  Keyboard.jack
  Math.jack
  Memory.jack
  Output.jack
  Screen.jack
  String.jack
  Sys.jack
```

```
Processing FILE
Base: Array
Type: jack
Files: 1
  Array.jack


Processing FILE
Base: NoSuchFile
Type: jack
Files: 0
```

In addition to the **`FileSet`** class, you are to implement a **`Util`** class that will contain a collection of class functions that we will add to as needed. The first function will be

`Util.getCommandLineArg(n)`

This function returns the nth command line argument, returning **`None`** if appropriate.

The `Scanner` class

The code for processing the files and preparing the report should be implemented in this class. The details of how you do this are up to you, but you might consider patterning your approach after the general outline of the **`FileSet`** class. You might also spend some time considering the approach that is encouraged by the class specifications for the Assembler (Section 6.3), the VMTranslator (Sections 7.3.3 and 8.3.3), the JackAnalyzer (Section 10.3), and the JackCompiler (Section 11.3). Look at the common theme that most of the specifications share and see if you can follow a similar flow.

The `os` module

Python provides a module named **`os`** that has many useful functions in it. For the FileSet class, the following ones are of particular interest: **`path.splitext()`**, **`path.isfile()`**, **`path.isdir()`**, **`chdir()`**, and **`listdir()`**. Other functions in this module allow you to get the current directory and to build up file specifications using the proper path separator for the operating system in use.

The `sys` module

This module is useful for system-related activites, including getting the command line arguments. The class variable **`sys.argv`** is a sequence containing all of the elements of the parsed command line.

Reading text files on Windows and *nix

Text files on Windows machines use separate carriage return and line feed characters while *nix machines only use the linefeed character. This can cause problems when processing a file on one machine that might have been written on the other. Since this is common place, Python has a file mode specifically intended to be able to read either type of file transparently. This mode is **`"rU"`** and is used as the second argument to the **`open()`** function following the filename.

Report Format

If no command line flag is provided, default to producing a report in the format represented by the following example.

```
============================================================
File Name
------------------------------------------------------------
File1.jack
File2.jack
File3.jack
------------------------------------------------------------
FILES: 3
============================================================
```

If the level is 1, then the report should include the file size as follows:

```
============================================================
File Name          Size (Bytes)
------------------------------------------------------------
File1.jack               2047
File2.jack              12345
File3.jack                603
------------------------------------------------------------
FILES: 3                14995
============================================================
```

If the level is 2, then the report should include the number of lines.

```
============================================================
File Name                  Lines
------------------------------------------------------------
File1.jack                    47
File2.jack                    45
File3.jack                     3
------------------------------------------------------------
FILES: 3                      95
============================================================
```

If the level is 3, then the report should include both the file size and the number of lines.

```
============================================================
File Name           Size (Bytes)             Lines
------------------------------------------------------------
File1.jack                  2047                47
File2.jack                 12345                45
File3.jack                   603                 3
------------------------------------------------------------
FILES: 3                   14995                95
============================================================
```

The details of the format, such as how long the dividing lines are (not to exceed 75 characters) and how much white space is between columns, are up to you, but the columns containing values that are summed should be right-aligned and must be in the left-to-right order shown. Furthermore, the dividing lines must be as shown in that the first and last are composed of equals signs while the inner two are composed of hyphens and are located directly above and below the list of files. If there are no files, then these two lines should be right next to each other.