COMP 112 - Lab 5

Due: 11:59pm, Friday December 5

Handin: place your `lab5.py` file in the folder `~/comp112/handin/lab5/`. Remember to issue the permissions command after you hand in the file:

```
setfacl -R -m u:rthorndy:rwx comp112/handin
setfacl -R -m m:rwx comp112/handin
```

Sample Files

- Sample Input File
- Sample Output File

Objectives

1. Work with the Python "Dictionary" type
2. Work with file input & output
3. Work with finicky output formatting

This assignment is just a single standalone program. Unlike the previous labs, you will not implement several smaller, distinct, functions. Instead, you are writing a single application that will run without any user intervention.

Your program is to take two command-line parameters. The first is the filename (and path, if not in the same directory as the program) of a file containing multiple address book entries. The second is the filename (and path, if not the same directory as the program) of an output file that will contain a sorted, formatted copy of the address book.

The Input File

The input file will be a text file with a single address book record per line, with fields separated by commas.

Your program must support any fields the input file may have it in. The name of the field will be given, followed by an equal sign ("="), followed by the value of that field for the record.

Only the "name" field is guaranteed to be present. There may be others, or just the name, but at the very least the name field will always be there.

Here are some example lines from an input file:

```
name=Rob,email=rthorndy@camosun.bc.ca,address=123 Thorndyke Lane
phone=250-555-1234, name=Joe, email = joe@joe.com
name=Jane
address=456 CompTech Rocks Ave, phone =555-0000,name=Bob
```

For simplicity, you may assume there is no “trickery” with the inputs. Every line *will* have a name field, and there will be no weird cases with newline characters or commas or equal signs as part of the field value that could create complications. However, there *can* be whitespace around the commas and equal signs; your program must be able to ignore these appropriately (although spaces *within* the text of the field values should be preserved).

You also don’t know the *order* of the fields; as seen in the example above, the fields may appear in any order within a single record.

The Output File

The purpose of the output file is to present a nice, formatted, sorted view of the address book.

The first line of the file should contain the field names that were found in the original input file. Even though you need to accept any field names, there should only be 5 or 6 that actually show up in the input file; your first line lists these 5 or 6 unique field names as headers, properly spaced out (see below).

The order of the field headings must be alphabetical, with the following exceptions:

- “name” must be the first heading
- “address” - if present - must be the last heading.

Each subsequent line represents an address book record, with the field values nicely lined up below the proper headings. If a record doesn’t contain a value for a given heading, that’s fine -- just leave it blank for that record.

Columns should be sized according to the maximum width they require based on the data in the input file. Specifically, as you’re reading the data you should be keeping track of the largest text value required for *each* field, so by the end, you know the size each column needs to be. Make the column size equal to the maximum text width for that field, plus 2 spaces for padding.

All values should be *right-justified*, except the address field (if present), which is the last one on the line, and should be left-justified.

The names should be listed in sorted order.

Given these specifications, the following would be the output file associated with the sample inputs given above:

```
name                     email          phone  address
 Bob                                255-555-0000  456 CompTech Rocks Ave
Jane
 Joe               joe@joe.com       555-1234
 Rob  rthorndy@camosun.bc.ca                    123 Thorndyke Lane
```

Note the left- and right-justifications, as well as the 2 spaces between each column.

Input Hints

Use `.split()` and `.strip()` when reading in the lines from the input file to break up the line into the relevant parts, and removing irrelevant spaces.

First use a `for..in` loop to go through every line in the input file; then, for each line, use `.split(",")` to create a list of the field name/value pairs. Now go through each name/value pair with another for..in loop, and for each name/value pair, using `.split("=")` to separate the field name and the field value, and use `.strip()` on each of these to remove any extraneous whitespace.

In that inner loop (the one separating the field name & value), check to see if:

1. you've seen that field name before, and if so,
2. if the value is the longest seen so far.

To do this, you should keep an extra dictionary used just for field names, to store the maximum width seen so far. Suppose you have just split the name/value pair so that you have the variables "name" and "value", and that you have a dictionary called fields that may or may not contain that field name:

1. Check to see if that field name has been seen before.
2. If so, check the maximum size so far, and compare to the size of value; update that maximum size if appropriate
3. If not, add the field name to the dictionary, and set the maximum size to the size of value.

It will look something like this:

```
if name in fields:
      if len(value) > fields[name]:
            # set the max size for this field to be len(value)
else:
      # add "name" as a key to the dictionary, set the value to be len(value)
```

Now, when the input file is done, you'll have a dictionary of field names along with their maximum widths ready to use for the output phase!

Output Hints

After the input file is done being processed, you should have two dictionaries available for use:

- `book`: all the data, with the keys being the names, and the values being a dictionary of the fields;

- `fields`: all the different fields that were seen, with the keys being the names of the fields, and the values being the maximum size each field needs.

The first thing you have to do is figure out the order the fields have to be displayed. "name" and "address" are treated specially, but the rest need to be sorted, and when you're doing your output, you have to use this sorted order for every record. It therefore makes sense to make a sorted list of field names before your start your output, so you can refer to this throughout the rest of your output process:

```
      sortedFields = sorted(fields)
```

First print the header row: print "name", followed by the sorted fields in order (ignored "name" or "address"), and ending with "address" (if it was present). For example

```
nameWidth = fields["name"]
headers = "%" + str(nameWidth) + "s" % "name"
for field in sortedFields:
      if field != "name" and field != "address":
            fieldWidth = fields[field]
            headers += "  %" + str(fieldWidth) + "s" % field
headers += "address"
```

Now you sort the address book, and then go through each record and print out the field values in a similar way:

```
sortedBook = sorted(book)
for name in sortedBook:
      data = book[name]
      nameWidth = fields["name"]
      outputLine = "%" + str(nameWidth) + "s" % name
      for field in sortedFields:
```

… ignore “name” and “address” fields

… check to see if “field” is available in this record’s “data”

… get the width required for this field, and either add empty spaces or

… the record’s value for that field using a “%” format string

… now handle the “address” field, if it is present in the record’s “data”