ECE 2524 - Spring 2016
HOMEWORK 6
Due before 11 p.m. on April 26

This assignment will take more time than previous assignments, and it is worth 80 points. In the final course average, it will be weighted twice as much as one of the previous homework assignments. Also, there is an extra-credit opportunity near the end.

Overview. You are to write several Python scripts that process text files in a Unix/Linux environment. It is acceptable to consult textbooks, Fedora utilities, and internet resources, but the work that you submit must be your own. This is not a "team" project, and you must not borrow or share code with another person. For full credit, your answers must be correct for this semester's version of Fedora, running the `bash` shell.

For each of the Python scripts that you submit, place lines near the beginning of your file similar to the following:

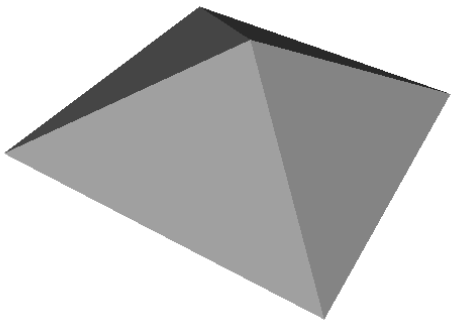
```
#! /usr/bin/python
#  ECE 2524 Homework 6 Problem 1  Jane Doe
```

Use the correct problem number and your own name, of course, in place of "Jane Doe". Insert a small number of comments into your code to explain your program to the grader. As usual, turn on execute permission for all of your scripts before submitting them.

More about PLY. PLY is the Stanford Triangle Format, and it is used to store representations of 3-dimensional shapes. The format is described at several places on the internet, including http://people.sc.fsu.edu/~jburkardt/data/ply/ply.html. You can display a valid PLY file on your PC using visualization tools such as Meshlab, which installs automatically on a Fedora system if you give permission when you try to run it by typing `meshlab`. A Windows version is available from http://sourceforge.net/projects/meshlab/files/latest/download.

An example PLY file is shown at the right. It describes the simple pyramid that appears below. The pyramid has 5 vertices, as indicated by the line "element vertex 5" in the file. Each vertex is specified by (*x, y, z*) coordinates, which are listed in the file immediately after the "end_header" line:

(0.0, 0.0, 0.0) ← index 0
(1.0, 0.0, 0.0) ← index 1
. . .
(0.5, 0.5, 0.3) ← index 4



```
ply
format ascii 1.0
comment file name: pyramid.ply
comment simple pyramid example
element vertex 5
property float32 x
property float32 y
property float32 z
element face 6
property list uint8 int32 vertex_index
end_header
0.0 0.0 0.0
1.0 0.0 0.0
1.0 1.0 0.0
0.0 1.0 0.0
0.5 0.5 0.3
3 0 1 2
3 0 2 3
3 0 1 4
3 1 2 4
3 2 3 4
3 3 0 4
```

The line “element face 6” indicates that 6 separate 3-D “faces” are specified by 6 lines of text immediately after the vertex list. Each of these face lines in the file begins with 3, which means that each “face” is a triangle for this example. (You may know that triangular meshes are very popular in the computer graphics community, and for 3-D modeling in general.) For each of these face lines of text, the next 3 integers represent indices for 3 vertices, with index numbers starting at 0 as indicated above. So, for example, the last line in the file specifies index values 3, 0, and 4, and therefore the corresponding triangle has 3-D coordinates (0.0, 1.0, 0.0), (0.0, 0.0, 0.0), and (0.5, 0.5, 0.3). Feel free to change some of the vertex coordinates, and then use Meshlab to see the effect on the pyramid.

Problem 1. (20 points) Write a Python script with file name `problem1.py` that verifies and prints information about a PLY file. In particular, the script should input a PLY file and report the number of vertices and faces. For example, the command

```
problem1.py pyramid.ply
```

should generate the following output lines to the standard output:

```
Number of vertices = 5
Number of faces = 6
```

Notice that these values should be taken from lines in the PLY file that begin with “element vertex” and “element face”, respectively. Do not assume that the “element vertex” and “element face” lines will always be the 5th and 9th lines in the file, because additional comment lines, etc. could be present.

Before reporting these values, your script should actually count the number of vertices and the number of faces that are given in the file. If any of these do not match the specified values from the “element” lines, then your script should output an error message similar to the following:

```
PLY format error:  5 vertices are specified, but file contains 6
```

This program should also check for other common problems, such as an invalid number of command-line arguments, or being unable to open a specified PLY file. In such cases, generate appropriate error messages, and exit the program.

Problem 2. (20 points) Write a Python script named `problem2.py` that changes the size of the object that is represented in a given ply file. A scale value should be provided as a command-line argument. For example, the command

```
problem2.py pyramid.ply 3.4 > pyramidout.ply
```

should create a new file named `pyramidout.ply` that is identical to input file `pyramid.ply` except that every vertex has been scaled (multiplied) by 3.4. For example, the input line `0.5 0.5 0.3` should be replaced in the output file by

```
1.7 1.7 1.02
```

Your `problem2.py` script is <u>not</u> required to check the number of vertices and faces as you did for Problem 1, although there is no harm in doing so. Your `problem2.py` script <u>should</u> verify that the correct number of command-line arguments is present, that the input file opened successfully, etc. For full credit on this problem, the grader must be able to open your output PLY file using Meshlab.

Problem 3. (40 points) Write an interactive Python script that can perform some aspects of inventory management. The program should read a text file containing information about electronic parts, and it should then modify the information by performing commands that are given in a second text file.

***Usage*.** Use the file name `problem3.py` for your program. It should be run using a Unix command line as follows:

`problem3.py` *inventory_data_file inventory_actions_file*

The 2 command-line arguments should be valid names of Unix text files. The formats of the file contents are given below. If an incorrect number of command-line arguments are given, or if the files cannot be opened for reading, then your program should generate an appropriate error message and exit gracefully.

Your program should input all of the information contained in the *inventory_data_file,* and then proceed to read and execute commands contained in the *inventory_actions_file*. After executing the last command from that file, your program should exit.

Inventory data file specification. An inventory data file begins with 2 header lines that your program should ignore. Every line of text after that is called a "record", and will conform to the following format:

<item ID><tab><description><tab><spaces><quantity 1 price><tab><spaces>
<quantity 10 price><tab><spaces><floor stock><tab><spaces>
<warehouse stock><newline>

There are 6 “data fields” per line, and they are separated by white-space character(s). In each case, a tab character is present between fields, and this could make it somewhat easier to parse the item descriptions. When <*spaces*> is specified, this means that 0 or more ASCII space characters may be present for alignment purposes. An example data file is shown below:

```
ItemID  Description                      Qty 1   Qty 10   Floor    Whse
-------------------------------------Price----Price---Stock---Stock
C86932  CAP,CHIP,TANT,47uF,16V,20%        1.09     0.96     470    8171
C86933  CAP,CHIP,TANT,1mF,16V,20%         0.39     0.35      59     500
C87118  CAP,CHIP,TANT,100uF,10V,20%       0.99     0.87     201    4010
R67811  RESISTR NET,10PIN,330 OHM,2%      0.19     0.17     105   12112
R67817  RESISTR NET,10PIN,1K OHM,2%       0.19     0.19      97   15410
R67843  RESISTR NET,10PIN,10K OHM,2%      0.19     0.17     114    8711
R90736  RESISTOR ASST,1/2W,5%            10.95     9.57      15      70
R90739  RESISTOR ASST,1/4W,5%             8.95     7.95      32      30
R90740  RESISTOR ASST,1/8W,5%            12.95    11.95       0      16
T48979  IC,7400 QUAD NAND                 0.59     0.47      71    9767
T49040  IC,7404 HEX INVERTER              0.59     0.48      55   10771
T49046  IC,7408 QUAD AND                  0.59     0.47      37    8110
```

The fields of each record are as follows:

- <*item ID*> This is a 6-character string. It serves as a unique identifier for a particular type of product that is for sale.
- <*description*> This is a string of no more than 30 characters, possibly containing space characters (but not tab characters).
- <*quantity 1 price*> This is a positive decimal value, in units of U.S. dollars. If the price is less than $1.00, a leading "0" will be present before the decimal point.
- <*quantity 10 price*> This is a positive decimal value, in units of U.S. dollars. If the price is less than $1.00, a leading "0" will be present before the decimal point.
- <*floor stock*> This is a nonnegative integer, representing the number of items for sale in a particular store.
- <*warehouse stock*> This is a nonnegative integer, representing the number of items available in a central warehouse.

You may assume that an inventory data file will never contain 2 different inventory records with the same <*item ID*>. Each of the other fields may be duplicated within the database. Note that the input data is not required to be perfectly aligned in vertical columns, even if it follows the specified format. Your code should assume that the inventory data file can contain any number of records. As usual in a Unix system, each line of text (including the last) ends with a <*newline*> character. You may assume that there will be no blank lines in the inventory data file.

Internal inventory representation. At start-up, your program should read from the inventory data file and store all of the database information in one or more Python dictionaries. You could use the `ItemID` field as a key, for example. This will simplify the program, especially if you use Python's advanced sorting capabilities.

Assume that the initial inventory list will be given in arbitrary order, although the example above happens to be sorted by <*item ID*>.

***Actions file specification*.** An inventory actions file will contain a sequence of commands for your program to identify and execute. Each line of text will specify one of the commands given below. Each command takes a fixed number of arguments. Your program may assume that each command name in the file is valid, and has the correct number of arguments. Each line in the file, including the last, is terminated by a <*newline*> character.

- `dump`
  This causes the complete inventory list to be sent to the standard output. More details about the output format are given below. The output should be sorted according to the most recent `sort` command (see below). If no `sort` command has yet been encountered, then by default the inventory items that are output should be sorted by <*item ID*>.

- `sort`<*tab*><*field specifier*><*newline*>
  This causes the inventory list to be sorted into ascending order by the specified field. The <*field specifier*> must be one of these strings: `ItemID` or `Description` or `Quantity1` or `Quantity10` or `Floor` or `Warehouse`. Sorting of the first 2 fields should be performed alphabetically, and sorting of the remaining fields should be performed numerically.

- `add`<*tab*><*item ID*><*tab*><*description*><*tab*><*quantity 1 price*><*tab*><*quantity 10 price*><*tab*><*floor stock*><*tab*><*warehouse stock*><*newline*>
  This causes the insertion of a new inventory record into the database list. Note that the "add" command with its arguments should be given on one line of text. The specification above appears on 2 lines just because of its length.

- `del`<*tab*><*item ID*><*newline*>
  This command causes the deletion of the inventory record for the indicated <*item ID*> from the internal list. If the specified <*item ID*> is not in the list, then the program should not print an error message, but simply proceed to the next command in the actions file.

There is no limit on the number of commands that can be present in the actions file. Any command can be repeated any number of times. A small example actions file is shown here:

```
sort  ItemID
dump
sort  Description
dump
del   R90736
del   C86932
add   R67825  RESISTR NET,10PIN,100K OHM,5%     0.20  0.18  200   9712
dump
```

***Output format*.** Normally, the only output from your program should be responses to dump commands, and this should be sent to the standard output. An example of correct output is given on below, for the data file and actions file given above. Your program should not generate additional output.

The first line of output from the dump command should be the title "`ECE 2524 Inventory Management System`". The second line should contain field labels, as shown in the example on the next page. Next, lines should be output containing the inventory data from the current internal list, aligned under the appropriate field labels. The next-to-last line is a statement of the number of records currently in the internal list. The last line of output for the dump command should be a row of hyphens.

```
ECE 2524 Inventory Management System
ItemID  Description                     Qty 1   Qty 10   Floor    Whse
C86932  CAP,CHIP,TANT,47uF,16V,20%       1.09     0.96     470    8171
C86933  CAP,CHIP,TANT,1mF,16V,20%        0.39     0.35      59     500
C87118  CAP,CHIP,TANT,100uF,10V,20%      0.99     0.87     201    4010
R67811  RESISTR NET,10PIN,330 OHM,2%     0.19     0.17     105   12112
R67817  RESISTR NET,10PIN,1K OHM,2%      0.19     0.19      97   15410
R67843  RESISTR NET,10PIN,10K OHM,2%     0.19     0.17     114    8711
R90736  RESISTOR ASST,1/2W,5%           10.95     9.57      15      70
R90739  RESISTOR ASST,1/4W,5%            8.95     7.95      32      30
R90740  RESISTOR ASST,1/8W,5%           12.95    11.95       0      16
T48979  IC,7400 QUAD NAND                0.59     0.47      71    9767
T49040  IC,7404 HEX INVERTER             0.59     0.48      55   10771
T49046  IC,7408 QUAD AND                 0.59     0.47      37    8110
The database contains 12 items.
------------------------------------------------------------------------
ECE 2524 Inventory Management System
ItemID  Description                     Qty 1   Qty 10   Floor    Whse
C87118  CAP,CHIP,TANT,100uF,10V,20%      0.99     0.87     201    4010
C86933  CAP,CHIP,TANT,1mF,16V,20%        0.39     0.35      59     500
C86932  CAP,CHIP,TANT,47uF,16V,20%       1.09     0.96     470    8171
T48979  IC,7400 QUAD NAND                0.59     0.47      71    9767
T49040  IC,7404 HEX INVERTER             0.59     0.48      55   10771
T49046  IC,7408 QUAD AND                 0.59     0.47      37    8110
R90736  RESISTOR ASST,1/2W,5%           10.95     9.57      15      70
R90739  RESISTOR ASST,1/4W,5%            8.95     7.95      32      30
R90740  RESISTOR ASST,1/8W,5%           12.95    11.95       0      16
R67843  RESISTR NET,10PIN,10K OHM,2%     0.19     0.17     114    8711
R67817  RESISTR NET,10PIN,1K OHM,2%      0.19     0.19      97   15410
R67811  RESISTR NET,10PIN,330 OHM,2%     0.19     0.17     105   12112
The database contains 12 items.
------------------------------------------------------------------------
ECE 2524 Inventory Management System
ItemID  Description                     Qty 1   Qty 10   Floor    Whse
C87118  CAP,CHIP,TANT,100uF,10V,20%      0.99     0.87     201    4010
C86933  CAP,CHIP,TANT,1mF,16V,20%        0.39     0.35      59     500
T48979  IC,7400 QUAD NAND                0.59     0.47      71    9767
T49040  IC,7404 HEX INVERTER             0.59     0.48      55   10771
T49046  IC,7408 QUAD AND                 0.59     0.47      37    8110
R90739  RESISTOR ASST,1/4W,5%            8.95     7.95      32      30
R90740  RESISTOR ASST,1/8W,5%           12.95    11.95       0      16
R67825  RESISTR NET,10PIN,100K OHM,2%    0.20     0.18     200    9712
R67843  RESISTR NET,10PIN,10K OHM,2%     0.19     0.17     114    8711
R67817  RESISTR NET,10PIN,1K OHM,2%      0.19     0.19      97   15410
R67811  RESISTR NET,10PIN,330 OHM,2%     0.19     0.17     105   12112
The database contains 11 items.
------------------------------------------------------------------------
```

You must use the same ordering of columns as shown here. Ideally, your program should align the output fields neatly into columns. You do not need to provide the exact spacing that is shown in the example below, but you should try to align the columns. The first 2 fields should be left-justified, and the last 4 fields should be right-justified. Your program should use spaces, not tabs, to do the alignment.

Output the monetary fields as dollar values using 2 decimal places. For convenience, it may be better to represent the values internally in units of cents, not dollars.

Problem 4: extra-credit opportunity (<u>not</u> required). For up to 20% extra credit on this assignment, create a Python script named `problem4.py` that updates a PLY description in creative ways. For example, it is possible to add color information to each face in the triangular mesh, and you could write a script to do that in an interesting way. Another example would be to subdivide each triangle in the mesh it into several smaller triangles, and possibly move the new vertex locations slightly to give the object a more textured appearance. If you decide to do this work, you will need to investigate the PLY format further. Use your imagination!

To run your script, the suggested command-line format is one similar to the format used for problem 2. The grader must be able to open any new PLY files that you create using Meshlab. If you do any work for extra credit, you must also submit a short, clear written description that explains what you have done. Describe how to run your program, and submit example input and output files that you have used to test your script.

How to submit your work for this assignment:

1. Verify that all of your files have the correct names, that they work correctly in the Fedora 23 environment, and that they contain the proper comment lines near the top.

2. Create a "tar" file by typing a command similar to the following:
   `tar cvf hw6_`*`name`*`.tar *.py`
   where *name* should be your last name (family name). If you do any extra-credit work, those files should also be included in the tar file.

3. Upload your tar file to Canvas before the deadline.

4. Test the tar file that you uploaded, to make sure it is correct. To do this, download it from Canvas to your Fedora machine. Place the tar file into a new directory that you create. In that new directory, type the following command to extract all of the original files:
   `tar xvf hw6_`*`name`*`.tar`
   Copies of your original files should appear in the new directory. Examine all of them to make sure that your submission is correct and complete. The grader will follow a procedure that is very similar to this. Notice that the grader will expect the exact file names that are specified in the assignment. To test your work, the grader will use files similar to the examples given in this assignment, as well as other input files that are much larger.
