**HOMEWORK:**

A “bag” is a data structure that handles duplicate items with a counter rather than a new entry. The UML for a “bag node” might look like this. We’ll keep it simple and make it a struct.

|  |
| --- |
| BagNode |
| + BagNode \* : next  + dataValue: string  + dataCount: int |

The first time an instance of a string is inserted, dataValue contains the string and dataCount is set to one. If another instance of that same string is inserted, dataCount is incremented. The “remove” operation decrements the counter until there is only one left; then on a subsequent remove the entire node is removed (or, as actual removal is expensive simply set the count to zero). The traversal displays both the string and the count if it is greater than one; for example

Adam

betty (2)

Diane (4)

fred

Ordering strings produces another issue; the relational operators use ASCII values so

betty < Diane

is false when it should be true. You’ll need to write a function that performs lexicographic comparison of strings, as we desire our traversal to be in lexicographical order.

Your program should perform the following tasks:

* ~~Display a friendly greeting to the user~~
* ~~Prompt the user for the name of a file that contains whitespace-delimited text~~
* ~~Accept that file name and attempt to open the file~~
  + ~~If the file fails to open, display an appropriate error message and exit~~
* ~~Process the file by~~
  + ~~reading the next word in the file~~
  + ~~removing any leading or trailing punctuation~~
  + ~~inserting the remaining word into the bag~~
* ~~Close the file~~
* ~~Prompt the user for another file name, for output~~
* ~~Accept that file name and open the file~~
* ~~Dump the traversal of the bag into that file and close the file~~

Your program should accept the names of the files as command-line parameters. If two file names are given, process the first and dump the traversal into the second. If only *one* file name is given, process the file and dump the traversal to the console. If no command-line parameters are given, prompt for both as described above.

The instructor will run your program against both small and large flat-text files, such as “Moby Dick” or “The Oxford English Dictionary.” Try it with the short story “Bottle Party” by John Collier (flat text available online) and your program might crash in a surprising way. Why? How would you fix it?