Programming Assignment: Word Counter

1. The problem

You have been hired by Books-For-Birds Publishing to help authors with their prose. You are supposed to warn authors if they use words too frequently. Authors give you a document as an ordinary text file. You have to collect the words of the file (case is significant; discard punctuation and spacing) and print two outputs. Both outputs list all the words and the number of times they appear, one word per line, no word duplicated. For example, you might print

```
example 1
the 18
they 2
```

The first output should be sorted by word (in alphabetical order), the second by number of occurrences (in reverse numerical order; ties can be in any order).

2. What to do

Your program should accept the text file from standard input. Output both lists to standard output, with a single blank line between the lists, and no other information (such as headings). Each line holds a single word and its count, separated by a single space.

You must use a hash table to store the counts. You may use any hash function and any collision-resolution scheme you wish.

You must write your own sorting algorithm to sort the output and your own hash-table routines. You may not use any library routines for sorting or hashing. You will lose 2 points if you use selection sort or insertion sort; you will get full credit for heapsort or quicksort. You may not use any other sorting method.

3. What to hand in

Your submission should include your program, all documentation, your program's output on the data in http://www.cs.uky.edu/~raphael/courses/CS315/data.hashing.txt, your own test data, and your program's output on that test data. As usual, submit a *tar*, *zip*, or other archive file to https://www.cs.uky.edu/csportal/.

4. Extra credit ideas

1. Implement both heapsort and quicksort and measure the number of data motions and comparisons each requires. (Maximum 0.5 extra-credit points)
2. Suppress very common words such as "the" from consideration by referring to a separate hash table of common words. (Maximum 0.3 extra-credit points)
3. Make your code Unicode-compliant, reading and writing in UTF-8, properly identifying letters (as

opposed to punctuation and spaces) in the entire Unicode space.
4. Implement several collision-resolution schemes and measure the number of key comparisons your program uses for each. (Maximum 0.5 extra-credit points)