CSC 316 – Data Structures

Programming Project #4 – Hash Tables and Spell Checking
Due Date: November 28, 2017, 11:59pm

Project Objectives
In this task, you will develop a spelling checker using a hash table. The assignment has two parts:

1. building the hash table, and
2. scanning the text for spelling errors using the hash table.

Description
Studies have shown that the 25,000 most common English words account for more than 99% of all written text; indeed, this is true not just in English, but in other languages as well. You are to design a hash table for an English spelling checker. When the program is run, it will first read in a file of the 25,144 most common English words. These words will be entered into the hash table. Then, the user's document will be read word by word. Each word will be searched for in the hash table, and flagged as a possible spelling error if it is not found.

Input
The word list of valid spellings is the file:

`http://courses.ncsu.edu/csc316/lec/001/wrap/Project4/dict.txt`

Your program must prompt the user for the name of the dictionary file containing the word list, the name of the text file to be spell checked, and the name of the output file.

Hash Table Construction
The dictionary contains 25,144 words. Even though the list is already in alphabetical order, using binary search to find a word would be quite slow, requiring 14-15 probes on average. With a hash table you should be able to reduce this to fewer than 3.

During the first phase, your program must read each word in the dictionary and store it in a hash table. Your hash table should be large enough to make searching fast, but not wastefully large. The size should also obey any constraints imposed by your collision resolution strategy.

Hash Function
You must select an appropriate hash function to build and access the table. Since the keys are strings of characters, a function operating on the `ORD` values of the characters is a good choice. A good hash function is simple, fast, and keeps the number of collisions relatively small.

Collision Resolution
You must also implement an effective collision resolution method. Many methods have been devised, and a few have been described in the textbook and in class. Any of these is acceptable, but you should feel free to explore others, such as those described in Chapter 3 of the book *File Organization and Processing* by A. L. Tharp (Wiley, New York, 1988).

Checking the Text
After reading the word file and building the hash table, the program will switch to its spelling-checking mode. The program will read the user's text file word by word, checking the spelling of each word. If a word appears to be mispelled, the program must write it out.

As your program searches for each text word, it should count the number of probes in the hash table. For the purposes of counting, a probe occurs whenever a text word is compared to a word in the table. A probe does not occur when a text word is compared to an empty table entry. Your program should print out the total number of probes it made during the spelling-checking phase before it terminates.

The dictionary does not contain every English word. It contains *understand*, but not *understands* or *understanding*. You are expected to strip off certain common suffixes and try again if the word is not found in its original form. Always search for the whole word first. If it is not found, apply the following rules:

- If the first letter of the word is capitalized, down-shift the first letter and try again; this helps to find words like *The* (e.g., words starting a new sentence).
- If the word ends in "'s", drop the "'s" and try again; this helps to find words like *cook's*.
- If the word ends in "s", drop the "s" and try again; this helps to find words like *cakes*. If it ends in "es", drop the "es" and search a third time; this helps to find *dishes*.
- If the word ends in "ed", drop the "ed" and try again *(cooked)*; if the word is not found, then drop only the "d" and try a third time *(baked)*.
- If the word ends in "er", drop the "er" and try again *(cooker)*; if the word is not found, then drop only the "r" and try a third time *(baker)*.
- If the word ends in "ing", drop the "ing" and try again *(cooking)*; then replace "ing" with "e" and try a third time *(baking)*.
- If the word ends in "ly", drop the "ly" and try again *(deliciously)*.

Finally, you should regard the apostrophe as a letter. Words like *don't* should not be broken into two pieces in either phase of the program.

These rules don't sound too bad, but remember that you'd have to apply three different rules to reduce *Bakers* or *Baker's* to the dictionary entry *bake*.

This list of rules is certainly not complete. You should expect some correctly spelled words to be flagged as misspelled by your program. And, you should expect some incorrectly spelled words

(bakeer) to be missed. (The Unix `spell` program suffers from the same problems, but is still very useful.)

Output
The program will not generate any output when building the hash table. During the checking phase, it should output any word whose spelling it deems to be questionable. It should also report: (1) the number of words in the dictionary, (2) the number of words in the text to be spell-checked, (3) the number of misspelled words in the text, (4) the total number of probes during the checking phase, (5) the average number of probes per word (of the original text file) checked, and (6) the average number of probes per lookup operation (note that a single word may require multiple lookup operations, per the above rules).

Submission
Submit your program using the **submit** tools, by 11:59pm on November 28, 2017. You must also submit a report describing your implementation, i.e., the hash table, the hash function, and the collision resolution method you used.

Grading

Hash table:	30 Points
Hash function:	30 Points
Collision resolution:	30 Points
Output (misspelled words, probe numbers):	10 Points
	100 Points

Important reminder: Homework is an individual, not a group, project. Students may discuss approaches to problems together, but each student should write up and fully understand his/her own solution. Students may be asked to explain solutions orally if necessary.