Scott Shrout

Programming Assignment 4

Due 11/9/2017

Abstract

In this assignment, we are tasked with utilizing a file reading, string parsing, and list traversal functions to perform a "spell check" of a provided document. To begin, we are required to read an input dictionary file containing a list of valid words. As each word is read, the first letter of the word is determined and the word is added to a list (of custom type MyLinkedList) corresponding with that letter. This is fairly straightforward, only requiring basic scanner functions to open the file and read each line token, a string function to determine the first character, and a function associated with the custom list classes (from Lab 6) to add the new word. As the dictionary would be static between SpellCheck objects (which I use to represent individual documents requiring a spell check), it is defined as a static array of MyLinkedList objects.

After the dictionary file is read in, we are now able to read the input file requiring a spell check. While there are a few options in this regard, I utilized the simple Scanner type along with its .next() method to read each word with the assumption that words are delimited by spaces. While this assumption is true in the majority of cases, there are some instances in the document where this turns out to not be so (for example, in the instance of words joined by a hyphen or comma with no proceeding space). Thus, for a spell checker utilized for a more serious purpose, one would likely need to consider more sophisticated string parsing techniques. For the purpose of this assignment, this seems to suffice.

As each word is read using the .next() method, it is stripped of a some characters to allow more words to match a dictionary entry. Specifically, hyphens, commas, semi-colons, colons, question marks, periods, and exclamation points are removed. The first letter of the word is then determined and an overloaded contains method is called, passing the lowercase of the word along with a small integer array that allows stats to be collected regarding the number of operations required. As the integer array is passed by reference, updates made by the contains method are reflected in the passed array. After traversing the appropriate list, the contains method returns a boolean value indicating whether the word was found in the dictionary list. The runSpellCheck function also makes an update to the integer array, updating counters related to words/not found (though this could be done in the called function as well).

After each word token has been checked against the dictionary entries, the spellCheck function displays stats including the words found, words not found, and the average number of comparison operations required for each.

Such a spell checking operation is quite taxing from an efficiency perspective. Though maintain a separate list for each letter of the alphabet reduces time complexity significantly, there are still an average of 3554 operations needed for found words and 7438 operations for words that are not found (in which every item in the list for a letter must be traversed). This complexity would be significantly greater for a realistic dictionary, such as the Oxford English Dictionary which currently contains 171,476 words.

Although I hesitate to speculate on how most spell checking programs work "under the hood," they seem to be able to apply spell checking to a document fairly efficiently. I think there are likely numerous "enhancements" that could made to the code in this assignment to at least bring us to the more ideal. For example, a separate list could possibly be maintained of words already found in the document that have been found to be valid. This would likely add value as

it is apparent many words in the document (and likely most documents) are used multiple times, requiring a traversal of the dictionary list upon every instance.