ECE242 Project 3
Searching and Sorting

We are all familiar with the spell checkers in our word processing software that read every word we type and complain when they see something they do not like. Now, you will have the chance to write your own spell checker, and in the process gain some appreciation for the task and maybe learn to be a little more forgiving when you are told you have misspelled your own name, again.

For this project, you will implement a program that performs a spell check on a text document you supply. This will not be as polished as the spell checkers you are likely used to, but it will give you some more experience with sorting and searching with a real life example. Let's start with some background before going into specifics.

You will be supplied with a text file, "worddb.txt". It contains over 50,000 English words (which really is not all that many, as you will see). Unfortunately, one night in a freak Perl accident, the order of words in the list was randomized! This will constrain your search abilities, something we will address later in the project. Luckily, however, the supplier of the word database was kind enough to put the number of words (not the number of lines, mind you) on the top line of the text file. This should make your lives a little easier when it comes to reading in and searching.

Each word will be read in turn from the file and placed into an array. This large array will be the accepted lexicon for this project; any word not on this list will be targeted by your spell check as an incorrectly spelled word. You will also be supplied with a sample document to spell check. Called "letter.txt", it is very simply written and formatted so that you can spend your time working with data structures and not going crazy over every special case that arises in writing. Please read this text file you will be spell checking **before you start the project**, as it has some useful information in it.

The details of the implementation are up to you (assuming you follow the guidelines which will be made clear soon) but you will want to read in a line of text from your sample document and check each word to see if it exists in your dictionary. If it does, all is well. If the word cannot be found in the dictionary, it will be flagged as a potentially misspelled word and handled as such.

The following are guidelines as to how the files in the project should be structured. The class and method names should match what is shown below, but feel free to add methods if necessary.

<u>Part A – Reading in the dictionary</u>

1. Download the input files you will need and save them in your project directory.
2. Create a class called **Dictionary**. This class will contain the following method:
   a. **ReadDictionary** – When invoked, this method opens the word database file, reads the number of words (the first line) and stores the rest of the words in an array of Strings whose size is equal to the number of words in the database. This array will be the dictionary you will search against.

Part B – SimpleSearch

1. Create a class called **SimpleDictionary** that ***extends*** **Dictionary** and has the method:
   a. **SimpleSearch** – This method takes a String as an argument and performs a **linear** search of the dictionary array for a match. If a match is found, return true. If no match is found, return false.

Part C – FastSearch

1. Construct another class called **FastDictionary** that ***extends*** **Dictionary**. Include the following methods:
   a. **OrderIndex** – This method puts the words in your dictionary in alphabetical order. **You must use the recursive version of MergeSort to sort the dictionary**. You are not allowed to use any of the standard sort methods available in the Java libraries!
   b. **FastSearch** – Unlike SimpleSearch, this method will perform a **binary** search of the dictionary array for a match. Keep in mind that you must use OrderIndex to sort the dictionary array before you can use FastSearch. As before, if a match is found, return true, else, false.

Part D – Spell Check File

1. Create a class called **SpellCheckFile**. This will be used to run a spell check on a given input text file and will contain the method:
   a. **CheckFile** – This method will be responsible for controlling the spell check on your document. It will take the filename of the document, as well as a Boolean field as to whether a FastSearch or a SimpleSearch is being performed. It will read the file line-by-line, split the line into words (removing any punctuation), and check each word against the dictionary using either FastSearch or SimpleSearch.

      If a given word is found in the dictionary, it should be printed to the console. If it is not found, it should be printed in parenthesis. This output will indicate which words may not be spelled properly. An example is below.

This:	Becomes:
good afternon, how are yuo?	good (afternon) how are (yuo)

      Note that the punctuation has been stripped away; this is fine.

Part E – Testing

Create a test class called **SpellChecker** which contains a main method. Do the following in the main method:

1. Create objects of classes **SpellCheckFile**, **SimpleDictionary**, and **FastDictionary**.

2. Read in dictionary using ReadDictionary from the SimpleDictionary object.
3. Perform spell check on letter.txt using the unordered dictionary and linear search
    a. Record how long the spell check takes, report in microseconds
4. Read in dictionary using ReadDictionary from the FastDictionary object.
5. Sort this dictionary using OrderIndex
6. Perform spell check on letter.txt using the sorted dictionary and binary search
    a. Record how long this spell check takes, report in microseconds. Do not include the time to sort the dictionary in your search time.

Part F – Going Further

Please place the answers to these questions as comments at the bottom of your SpellChecker class. Think carefully about these questions and answer them as specifically as possible.

1. Compare the search times for your two dictionaries. Using the results from your tests, how long do you estimate it would take to check the spelling in the same text file if your dictionary had 500,000 entries with :
    a. SimpleDictionary
    b. FastDictionary

    Report your answer using concrete times, not just " this many times longer"

2. This is a rather simple spell checking tool. Comment on the limitations of this approach in terms of:
    a. Quality of the spell check: What words, characters, formatting gives it trouble
    b. Dictionary size and flexibility in terms of the data structures used

3. Mention some changes that could be made to improve the handling of the word database to make it more flexible.