A Spell Checker Part 2

The homework problems for this chapter combines **the reading from the last three** chapters into a single, larger problem, where you write a **Spell Checker** program.

Read through the instructions here and complete the second half.

- The `spellCheck()` function checks the spelling of all words in a stream. It reads a word from the stream and checks whether it is contained in a list of correctly-spelled words. If the word is in a list of excluded (common) words, you should just ignore it.

A correct version of the `fileToWords()` function from **H17** has been included in the library linked to your code. You don't have to bring your own over.

The `spellCheck()` function returns a `vector` of `Word` structures. Each `Word` object in in the vector contains the misspelled word, along with a `vector` of the position in the file where the word was encountered.

The `Word` structure is defined in the header file. For instance, if **"artcic"** (a misspelling of **arctic**) was found three times in the input stream, there would be one entry in the returned `vector`, and it might contain data like this:

```
word: artic
pos: [7, 95, 89]
```

Pseudocode

Here's the pseudocode for the function which takes an **istream&** and two **const vector<string>&** parameters (the **dictionary** and the **ignored** words):

```
in: input stream
results: vector
while in // not end of file
   current-position <- in.tellg()
   If current-position is -1 (at end of file) Then
       Exit the Loop
   word: string

   Read next word (in >> word >> ws)
   word <- clean(word) // lowercase, no punctuation

   Search results for word
   If word found Then add position to element in results
       Continue (next iteration)

   Search excluded words for word
   If word found Then continue (next iteration)

   Search the dictionary for word
   If word found (Not misspelled) Then continue

   Create a Word, populate with word, position
   Add new Word to results
End Loop
Return results (misspelled words and their positions)
```

You can do this with a series of **nested if else** statements, but the **continue** statement makes it quite a bit easier. The **continue** statement will jump back to the top of the loop, much like **break** jumps out of a loop.

You can use **continue** to avoid a series of **if (! found)** blocks in your code.

Use **make run**, **make test** and **make submit** as usual.

Ask questions on Piazza if you get stuck or come by my office hours.