CSCI 1300 Introduction to Computer Programming
Instructor: Knox
Assignment 7
Due Sunday, Nov 5, by 6:00 pm (5% bonus 6pm Fri Nov 3rd, 2% bonus 6pm Sat Nov 4th)

This assignment requires you to create four files: SpellChecker.h, SpellChecker.cpp, WordCounts.h and WordCounts.cpp. The class definitions should be in the .h files and the implementation of the methods in the .cpp files. Your main.cpp file should be used to test your implementation of your classes. You can create a project in CodeBlocks to create the main.cpp file and add the two classes to the project. CodeBlocks can create projects (use console application template) and will create the .h and the .cpp files when you add classes to the project. Code Blocks will compile each file for you and link them into the project executable for testing. You should ***NOT*** be using the **`#include "SpellChecker.cpp"`** and **`#include "WordCounts.cpp"`** in your main file to include your code into main.

1. Once you have your code running on your virtual machine (VM), submit a zip file with *main.cpp, SpellChecker.h, SpellChecker.cpp, WordCounts.h and WordCounts.cpp* to the autograder COG .
   https://web-cog-csci1300.cs.colorado.edu

2. You **must also** submit your code to Moodle to get full credit for the assignment.
   - 10 points: Comments at the top of your source files should include your name, recitation TA, and the assignment and problem number.
   - 10 points: Algorithm descriptions comments in your code submission to describe what your code is doing.
   - 10 points: Code in your **`main()`** function that is testing the functions in **`SpellChecker.cpp`** and **`WordCounts.cpp`**. 10 points will be deducted if the main.cpp you submit to moodle does not have code testing the functions you have written in **`SpellChecker.cpp`** and **`WordCounts.cpp`**. Remember that COG *does not* run your `main()`, it runs its own main. (Code you write in `main()` will not impact your COG runs).
   - NEW starting with this assignment! 10 points: Proper and consistent indentation of the code. You will have 10 points deducted if the indentation is missing or if the code in the files is using an inconsistent style.

Part I

In this part of the assignment, you are to create a class, ***SpellChecker***. You will define some class data members, member methods and helper functions. The class methods will be used to check the spelling of words and assess the word count across documents. Elements of this assignment are intentionally vague; at this point in the semester, you should be able to make your own decisions about appropriate data structures for storing and looking up data, as well as defining helper functions. You can assume that your code will never be storing more than 10,000 valid or corrected words.

SpellChecker should have **at least** the following *Public members*:

- *string language*: the name of the language this spell checker is using (i.e. "English", "Spanish", "Italian", "Hindi", ...)

SpellChecker should have **at least** the following *Private members*:

- *char start_marker*: used for marking the beginning of an unknown word in a string
- *char end_marker*: used for marking the end of an unknown word.

SpellChecker should have three constructors:

- Default Constructor, the one with no arguments.
- Second constructor that takes a string for the object's *language*.
- Third constructor that takes a string for the object's *language* and two filenames as parameters. The first filename specifies the file with correctly spelled words and the second filename specifies the misspelled words with their corrections.

You will be dealing with two different files:

- The data in the first filename supplies a list of correctly spelled words, one word per line:

```
aardvark
apple
acquire
. . .
```

- The data in the second filename contains a list of misspelled words and their correct spellings. The word and its correction are separated by a tab character ('\t'):

```
teh	the
todayy	today
tork	torque
. . .
```

It is **very** important you understand the format of this file. The correctly spelled words may have spaces in them! For example a file that converts common texting abbreviations into words:

```
btw    by the way
imnsho    in my not so humble opinion
lol    laugh out loud
ttyl    talk to you later
yam    yet another meeting
. . .
```

The constructor with the filename arguments should open the files and read them into an appropriate data members of the class. To find if a *word* is a valid spelling or is a misspelling, you should think about storing the words in the right structure so that it's easy to search and access it.

SpellChecker should also include the following public methods:

- **bool readValidWords(string filename)**: this method should read in a file in exactly the same way as detailed in the description of the constructor. This file will have the format specified for correctly spelled words. This method should return a boolean of whether or not the file was successfully read in. This method should add the words from the file to the list of words already contained in the object.

- **bool readCorrectedWords(string filename)***:* this method should read in a file in exactly the same way as detailed in the description of the constructor. The file will have the format specified for the wrongly spelled words and their corrected spellings. This method should return a boolean of whether or not the file was successfully read in. This method should add the words from the file to the list of words already contained in the object.

- Setters and Getters for the markers to be used for unknown words (see description of marker use below).
    - return *true* for the settters if the new marker has been accepted
        - bool setStartMarker(char begin)
        - bool setEndMarker(char end)
    - char getStartMarker()
    - char getEndMarker()

- **string repair(string sentence)**: *Repair* will take in a string of multiple words, strip out all the punctuation, ignore the case and return the sentence with all misspellings replaced or marked. For example: here are what the following calls would return:

```
repair("todayy")                    "today"
repair("todayy, is teh day!")        "today is the day"
```

  If you cannot find a word in the list of valid words or in the list of misspelled words (for instance, if the word is misspelled beyond recognition), you should just return the misspelled words with the `start_marker` in front and the `end_marker` at the end. For example: if `start_marker` and `end_marker` are both '~', the call:

```
repair("ahsjdklfha")                  "~ahsjdklfha~"
repair("tomor is another day!")       "~tomor~ is another day"
repair("Teh brown asdhf jumped.")     "The brown ~asdhf~ jumped"
```

- **(Challenge Problem) repairFile(string input_filename, string output_filename)**: *Repair* will process all the lines in the input file and write the repaired lines to the output file. The resulting file would still have all of the punctuation, but with individual words corrected. One way to break this down into simpler tasks would be to first check each stripped word for spelling. The second task would be to replace the corrected word within the sentence will all its punctuation.

Testing

Testing of your class and all of its methods is now in your hands. You must determine the test cases that will test if your implementation returns the correct results in all conditions. For example, you would need to write code that will declare SpellChecker objects with each of the possible constructors, to verify that each of those methods will create and initialize the object correctly. The same must be done for each of the other public methods to verify that your implementation works correctly in all possible conditions and ordering of calls to those methods.

Once you are satisfied that your code works as intended, submit it to COG for its evaluation.

Part II

In this part of the assignment, you are to create a class, ***WordCounts***. You will define some class data members, member methods and helper functions. The class methods will be used keep a running count of the number of times each word is being used. You can assume that there will never be more than 10,000 unique words being counted. Your class will provide the following public methods to support counting word usage:

- ***void tallyWords(string sentence):*** This function will take in a string of multiple words, remove the punctuation, and increment the counts for all words in the string. If a word is not already in the list, add it to the list. This function is used to keep **a running count** of each unique word processed; that means multiple calls to the function should update the count of the words, not replace them. If we call the function three times:

  ```
  resetTally();
  tallyWords("the brown fox.");
  tallyWords("the red fox.");
  tallyWords("teh blue cat.");
  ```

  The count for the words "the" and "fox" should be 2, the count for the words "brown", "red", "blue", "cat", and "teh" should be 1.

- **int getTally (string word)***:* return the current count of the given word. If the word is not found to be in the current list of words, return 0.

- **void resetTally()***:* reset all word counts to zero.

- **int mostTimes(string words[], int counts[], int n)***:* find the ***n*** most common words in the text that has been counted and return those words and counts in via the arrays given as parameters. Assume the arrays are large enough to hold the number of elements requested.

Testing

Testing of your class and all of its methods is now in your hands. You must determine the test cases that will test if your implementation returns the correct results in all conditions. For example, you would need to write code that will declare ***WordCounts*** objects with each of the possible constructors, to verify that each of those methods will create and initialize the object correctly. The same must be done for each of the other public methods to verify that your implementation works correctly in all possible conditions and ordering of calls to those methods.

Once you are satisfied that your code works as intended, submit it to COG for its evaluation.