CSCI 1300 Introduction to Computer Programming
Instructor: Knox
Assignment 7
Due Sunday, March 26, by 5:00 pm

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 will create the .h and the .cpp files for you and compile them into the project for testing. Or you can use the **`#include "SpellChecker.cpp"`** and **`#include "WordCounts.cpp"`** in your main file to use your classes in main.

Once you have your code running on your virtual machine (VM), you must zip the .h and the .cpp files into a single .zip file and submit that file to the autograder COG. COG will verify that calls to your class methods provide the correct behavior. It will not provide detailed descriptions of the data used to validate your code, only the test case description of what it was testing. You will need to provide test code to validate your implementation in your main.cpp file.

Submitting Your Code to Moodle:
You must also submit your code to Moodle to get full credit for the assignment, regardless of the score you receive from the auto-grader. Create another .zip file which includes main.cpp, SpellChecker.h, SpellChecker.cpp, WordCounts.h and WordCounts.cpp to be submitted to Moodle.

Comments at the top of your source files should include your name, recitation TA, and the assignment number. **Please also include comments in your code submission to describe what your code is doing. Provide a description of each method being implemented, details of what the method does, the parameters, and the return value. Also include a section about your collaborations, either classmates, CAs, or online resources used to complete the assignment (see the collaboration policy in the syllabus). TAs will be checking that your code has comments.**

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 and correct the spelling of words. 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 misspelled 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", ...)

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

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

SpellChecker should have three constructors (set the object's data members to some default values):

- Default Constructor, the one with no parameters.
- Second constructor that takes a string parameter 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 file types:

- 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 parameters should open the files and read them into the 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 loadValidWords(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 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 loadMisspelledWords(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).
    - **void setBeginMarker(char begin)**
    - **void setEndMarker(char end)**
    - **char getBeginMarker()**
    - **char getEndMarker()**

- **string fixUp(string sentence)***:* Fixup will take in a string of multiple words, break it into individual words, strip out all the punctuation, and ignoring the case, return the sentence with all misspellings corrected and unknown words marked (see below). For example: here are what the following calls would return:

```
fixUp("todayy")                  "today"
fixUp("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 `begin_mark` in front and the `end_mark` at the end. For example: if `begin_mark` and `end_mark` are both '~', the call:

```
fixup("ahsjdklfha")              "~ahsjdklfha~"
fixUp("tomor is another day!")   "~tomor~ is another day"
fixUp("Teh brown asdhf jumped.") "the brown ~asdhf~ jumped"
```

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 and 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 to keep a running count of the number of times each word is being used. All processing of words should be case insensitive. 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 countWords(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:

```
resetCounts();
countWords("the brown fox.");
countWords("The red fox.");
countWords("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 getCount(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 resetCounts()***:* reset all word counts to zero.

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

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.