CSCI 1300 - Intro to Computer Programming
Instructor: Fleming/Gupta
Homework 6

Due Sunday, March 11th, by 6 pm
+5% bonus if submitted by Friday March 9th 11:55 pm,
+2% bonus if submitted by Saturday March 10th 11:55 pm

This assignment is due **Sunday, March 11th 6pm**.

- ***All components*** **(Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by Sunday, March 11th 6:00 pm for your homework to receive points.**
- Complete submissions (Cloud9 workspace, moodle quiz attempts, and zip file) before **Friday March 9th 11:55 pm** will receive a 5% bonus, and complete submissions before **Saturday March 10th 11:55 pm** will receive a 2% bonus.

Develop in Cloud9: For this assignment, write and test your solution using Cloud9.

Submission: All three steps must be fully completed by the submission deadline for your homework to receive points. Partial submissions will not be graded.

1. ***Share your Cloud 9 workspace with your TA:*** Your recitation TA will review your code by going to your Cloud9 workspace. *TAs will check the last version that was saved before the submission deadline*.
    - Create a directory called **Hmwk6** and place all your file(s) for this assignment in this directory.
2. ***Submit to the Moodle Autograder:*** Head over to Moodle to the link **Hmwk 6**. You will find 3 programming quiz questions for each part of the assignment. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date.
3. ***Submit a zip file to Moodle:*** After you have completed all the questions and checked them on Moodle, ***you must submit a zip file with the .cpp file(s) you wrote in Cloud9***. Submit this file going to **Hmwk 6 (File Submission)** on moodle.

Style and Comments (26 points)
Comments (10 points):

- Your code should be well commented. Use comments to explain what you are doing, especially if you have a complex section of code. These comments are intended to

help other developers understand how your code works. These comments should begin with two backslashes (//).

- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Spring 2018
// Author:
// Recitation: 123 – Favorite TA
// Cloud9 Workspace Editor Link: https://ide.c9.io/…
// Homework 6 - Problem # …
```

NEW: Indentation (6 points):

- Your code should use some form of indentation. Please look at examples posted on moodle as well as the example below (**checkMpg)** if you are unsure of how to use indents.

Algorithm (10 points):

- Remember to include in comments, before the function definition, a description of the algorithm you used for that function.
- This is an example C++ solution to problem 5 of Homework 1. Look at the code and the algorithm description for an example of what is expected.

```
/**
 * Algorithm: that checks what range a given MPG falls into.
 *     1. Take the mpg value passed to the function.
       2. Check if it is greater than 50.
              If yes, then print "Nice job"
       3. If not, then check if it is greater than 25.
              If yes, then print "Not great, but okay."
       4. If not, then print "So bad, so very, very bad"
 * Parameters: miles per gallon (float type)
 * Output: different string based on three categories of
 *          MPG: 50+, 25-49, and less than 25.
 * Returns: nothing
 */

void checkMPG(float mpg)
{
       if(mpg > 50) // check if the input value is greater than 50
       {
              cout << "Nice job" << endl; // output message
       }
       else if(mpg > 25) //if not, check if is greater than 25
       {
              cout << "Not great, but okay." << endl; // output message
       }
       else // for all other values
       {
              cout << "So bad, so very, very bad" << endl; // output message
       }
```

```
}
```

The following algorithm description does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would not receive full credit.

```
/**
 * Checks mpg
 */
void checkMPG(float mpg) {
      if(mpg > 50) {
            cout << "Nice job" << endl;
      }
      else if(mpg > 25) {
                  cout << "Not great, but okay." << endl;
      }
      else {
                  cout << "So bad, so very, very bad" << endl;
      }
}
```

Test Cases (4 points)

Code compiles and runs (4 points):

- The zip file you submit to moodle should contain .cpp file(s) that can be compiled and run on Cloud9 with no errors. The functions should match those submitted to the autograder.

Background

For this assignment you will be creating a spellchecker program that will return words and phrases with the correct spelling if it is incorrect. You will be using your knowledge of arrays and file I/O for this homework.

Problem Set

Question 1

Write a **ReadCorrectWords** function to store all known, correct words for your spellchecker.

It should have the following parameters in the order provided:

- a string *filename*
- an array of strings *correctWords[]*
- the size of the array
- an integer *startIndex*.

Your function should return the total number of correct words in the array *correctWords*.

Your function should open the file identified by *filename* and read lines from the file line by line. The file will have the following format:

correctWords.txt

```
chicken
fish
cow
bubbles
pear
kiwi
```

You can expect exactly one word on each line. Each word is a correct spelling and should be added to the *correctWords* array. The parameter *startIndex* is the index at which you should begin adding your words.

If I called your function with the file above and *startIndex*= 0, your *correctWords* array should look like this:

```
["chicken", "fish", "cow", "bubbles", "pear", "kiwi"]
```

Your function should return 6 (for 6 words total in the array):

If I call your function *again* with *startIndex*=6 and the file below, your function should add the new words from the file to the array *correctWords* starting at position 6.

correctWordsPt2.txt

```
ocean
lake
mountain
```

Your final *correctWords* array would like like this:

```
["chicken", "fish", "cow", "bubbles", "pear", "kiwi", "ocean", "lake", "mountain"]
```

We will call your function multiple times with different files and the same array to make sure you append new words correctly!

Edge cases:

- if the file does not exist, return -1
- if the starting index of the array is greater than the size of the array, return -1

Question 2

Write a function **ReadMisspelledWords** to read in pairs of misspellings and correct spellings.

It should have the following parameters in the order provided:

- a string *filename*
- a 2D array of strings *misspelledWords[][2]*
- the number of rows in the array *misspelledWord*
- an integer *startIndex*.

Your function should return the total number of word pairs in the array *misspelledWords*.

Your function should open the file identified by *filename* and read lines from the file line by line. The file will have the following format:

misspelledWords.tsv

```
chickn chicken
chicke chicken
chcken chicken
fsh    fish
fishe  fish
bubbls bubbles
bubles bubbles
per    pear
peart  pear
```

The file will be a tab separated values (.tsv) file. You can identify tabs by the `\t` character. The first value on each line is a misspelled word, and the second value on each line is the correct spelling of that word. For example, on the first line `chickn` is followed by a `\t` and then the correct spelling `chicken`.

You should read these values into the 2D array so that the first column contains misspellings and the second column contains the correct spelling. For the file above and *startIndex* = 0, your array *misspelledWords* should look like this, and your function should return 9 (for 9 word pairs in the array):

```
[
 ["chickn", "chicken"],
 ["chicke", "chicken"],
 ["chcken", "chicken"],
 ["fsh", "fish"],
 ["fishe", "fish"],
 ["bubbls", "bubbles"],
 ["bubles", "bubbles"],
 ["per", "pear"],
 ["peart", "pear"]
]
```

startIndex identifies the row in the array at which you should begin to add the words in the file. If after calling the function a first time I called it again with the below file and *startIndex*= 9:

moreMisspelledWords.tsv

```
ocen   ocean
ocan   ocean
```

Then I would expect my array to look like this. Your function should return 11 for 11 word pairs in the array.

```
[
 [chickn, chicken],
 [chicke, chicken],
 [chcken, chicken],
 [fsh, fish],
 [fishe, fish],
 [bubbls, bubbles],
 [bubles, bubbles],
 [per, pear],
 [peart, pear],
 [ocen, ocean],
 [ocan, ocean]
]
```

We will call your function multiple times with different files and the same array to make sure you append new words correctly!

Edge cases:

- if the file does not exist, return -1
- if the starting row of the array is greater than the total number of rows in the array, return -1

Question 3

Write a function **CheckWord** to check if a word is spelled correctly.

It should have the following parameters in the order provided:

- a string *word*
- an array of strings *correctWords*
- Size of the array *correctWords*
- a 2D array of strings *misspelledWords[][2]*

- the number of rows in the *misspelledWords* array.

Your function should do the following with *word*:

1. Convert *word* to lowercase.
2. Check if the *word* is correctly spelled by seaching for it in *correctWords*. If it is found, the function should return the word itself.
3. If the word is not found in *correctWords*, check if it is a misspelling we know of by searching for it in *misspelledWords*. If it is found, return the correct spelling.
4. If the word is not found in either array, return the string `"unknown"`.

The return value should always be lowercase.

Question 4

Write a function **CheckPhrase** to check spelling in a phrase.

It should have the following parameters:

- a string *phrase*,
- a string *outputFile*
- an array of strings *correctWords*
- Size of the array *correctWords*
- a 2D array of strings *misspelledWords[][2]*
- the number of rows in the *misspelledWords* array.

You should do the following with *phrase*:

1. Separate the phrase into words.
2. Call CheckWord on each word to check the spelling of the word and construct a corrected phrase.
    - Note: the resulting phrase will contain “unknown” if any words are not found in the arrays
3. Open the *outputFile* in append mode.
4. Append the corrected phrase to the output file.

Edge Conditions

- Your program should print "*Phrase is empty. Nothing to write in file*" to the console if your phrase is empty. **Note:** Your program should not write anything to the output file in this case.
- Your program should print "Unable to create/ access output file" if you're not able to access your output file.

Example:

If I call **CheckPhrase** with the phrase `"chicken fish and bubbls"` , the arrays specified in the previous examples, and this output file:

outputFile.txt

```
ocean bubbles unknown pear
bubbles fish unknown unknown fish
```

then I would expect the output file to look like this after the function call.

outputFile.txt

```
ocean bubbles unknown pear
bubbles fish unknown unknown fish
chicken fish unknown bubbles
```

Question 5

Write a function **CheckFile** to check the spelling in an entire file.

It should have the following parameters in the order provided:

- a string *inputFile*
- A string *outputFile*
- an array of strings *correctWords*
- the size of the array *correctWords*
- a 2D array of strings *misspelledWords[][2]*

- the number of rows in the *misspelledWords* array.

Each line of the *inputFile* contains a single phrase. *outFile* will be created/opened in CheckPhrase function.

Your function should open the file and check the spelling of each line by calling **CheckPhrase**.

For example, if my *inputFile* was the one provided below:

testSpellChecker.txt

```
kiwi ocean bubbls
bubbles ocn chickn
lake
kiwi per
```

I would expect the following to be appended to my *outputFile*

```
unknown ocean unknown
unknown unknown unknown
lake
unknown per
```

Edge Conditions

- Your program should print "invalid" if you're not able to access your input file.

Submitting Your Code to the Autograder on Moodle

You must name the functions as indicated in each problem description. The cout formats provided for each problem are not suggestions – they *MUST* be followed precisely, word-for-word and including all punctuation marks, otherwise the autograder will not recognize your results and you will not receive credit.

If there are errors in your solution to a particular problem, a button labeled “Show differences” will appear below the table of tests after you hit “check”. This can be a very useful tool in helping you find small typos, especially in cout statements.
