Programming Project 1
EE312 Fall 2016

General: This project will give you a chance to dust off all the cobwebs on your programming skills. The project requires that you understand characters and strings, arrays, functions, loops and conditionals (all prerequisite knowledge). However, just because this is technically "a review", don't assume it's easy! You'll write a very simple spell checker. You must produce the output EXACTLY as specified in this document. There is some flexibility permitted in your solution to encourage you to adopt your own ideas for an algorithm. The limits to that flexibility are explained below. Any extra output, or output that is inconsistent with these requirements will result in lost points.

Your Mission: Edit the file "Project1.cpp". You must implement the function *spellCheck*. You may find it useful to write several other functions as well. In fact, I encourage you to avoid writing the whole project as one big (ugly) function. You will find big (ugly) functions are very difficult to debug, and we will find it very difficult to help you.

There is only one stage on this project, writing the *spellCheck* routine. The *spellCheck* function has two parameters. The first parameter (*article*[]) is a pointer to an array of characters. The contents of this array are an article that you need to spell check. The end of the article is marked with the normal `0` (marking the end of a string). The article includes punctuation, upper and lower case words, numbers, and abbreviations. Your function must print every word in the article that cannot be found in the *dictionary*. The *dictionary* is the second parameter to the function (more on this later).

For this project, a word is a sequence of two or more letters with no intervening punctuation, numbers or other non-letters. The character immediately preceding the first letter in the word must not be a letter, or it must be the beginning of the article. The character immediately following the last letter in the word must not be a letter, or it must be the end of the article. So for example if the article were, "The red firetruck went round a corner at 50mph", there would be eight words (the, red, firetruck, went, round, corner, at, mph). Note that "truck" doesn't count because it is embedded inside "firetruck" (the character immediately preceding 't' is a letter). Note that "a" is not a word because it is not at least two letters long. And finally, note that mph is a word even though the character preceding the 'm' is not a space. Words don't have to be preceded by a space, just by something other than a letter (or the start of the article, of course). These rules are intended to make finding words EASY. Just look for a letter, that marks the beginning, and keep scanning for consecutive letters. When you run out of letters, that marks the end.

For each word in the article, you must see if there is a matching word in the *dictionary*. The dictionary is also a character string. The end of the string is marked (as always) with a `0`. The end of each word is marked with a `'\n'`. A word in the dictionary can contain any character (other than the '\n' character). That means the dictionary can contain words like "don't" or "can't". Naturally, words from the dictionary that have non-letter characters in them (like "don't") will never match a word from the article. Such words can still appear in the dictionary. All words in the dictionary are sorted in alphabetical order.

For this project, a word in the article matches a word in the dictionary if the characters from the two words are the same except for the case of the letters. So "cAT" and "Cat" match each other, while "Id" and "I'd" do not match each other. For every word in the article, determine if there is a matching word in the dictionary. If there is a matching word, do nothing. If there is not a matching word in the dictionary, print the word from the article.

You can print the words in several ways, depending on how you store the words in memory. If the word is terminated with a 0 like a regular string, then you can use printf:

```
printf("%s", word);
```

If your strings are not terminated with the zero, then you will need to print them one letter at a time (using a loop). You can print a single letter in one of two ways:

```
printf("%c", letter);
```

or

```
putchar(letter);
```

When you're debugging your program, you will want to make sure that all the characters you have printed show up on the screen. That may sound silly, but it is a real problem. The computer does not always put characters on the screen, even if you've used *printf* or *putchar* to print them! If you want to make sure characters have been forced to the screen, print a new line ("\n") or use `fflush(stdout)`.

OUTPUT REQUIREMENTS: Your function **must only print the misspelled words from the article**. Each word that you print must be printed on a line by itself (i.e., you must print a '\n' after each word of output). You may print words in all lower-case, in all upper-case or in mixed case. The case (upper/lower/mixed) that you print the word does not need to match the case that is used in the article. Finally, you must only print words that (a) satisfy the definition of being a word for this project (see above) and (b) do not appear in the dictionary. If the same misspelled word appears more than once in the article, then you may print that word exactly one time, or you may print that word as many times as it appears in the article. Your program must produce no output other than the misspelled words (one word per line). Please keep in mind that we will test your program with different articles and different dictionaries than have been provided to you. Your program should work with any article or dictionary that satisfies the requirements listed above. Not all articles start with a letter or end with a letter. Not all articles have at least one word. Not all dictionaries have at least one word.

Guidance/Advice

We strongly recommend that you develop the project in phases. Your first goal should be to produce a working program that correctly reads (and prints) all the words from an article. Print the words regardless of whether the word is spelled correctly or spelled incorrectly. Don't just sketch pseudo-code of this phase – really write it, compile it, run it and debug it. Once you can discover and print each word from the article, proceed to the next phase of development where you begin consulting the dictionary for each word and printing the word(s) only when it is misspelled. It is a truism for EE312, if you can make the time to write each program twice (even if you don't build all the functionality into your first version of the program) you will learn more and learn more quickly than if you try to do the entire project at once.

Project 1 requires no knowledge of pointers. If you are very comfortable with pointers, feel free to use them. If you are not comfortable, then don't use them. The solution to the project has exactly zero occurrences of "*" in the .cpp file (no kidding). Don't use variable length arrays in your solution. Think about what size you want to make your arrays.

Finally, when we are considering the bonus point for Project 1, we are looking for a combination of good program design (good coding style, including comments, indentation and variable/function names as well as a concise implementation of the required functionality), and a robust implementation (your program works correctly under all reasonable sets of input). Good luck!

CHECKLIST – Did you remember to:

- ☐ Re-read the requirements after you finished your program to ensure that you meet all of them?
- ☐ Make sure that your program passes all our testcases?
- ☐ Make up your own testcases?
- ☐ Upload your solution to Canvas?
- ☐ Download your uploaded solution into a fresh directory and re-run all testcases?