CS 2043 – ASSIGNMENT #5

Assigned: **2/23/2023**
Due: **3/3/20223, by 11:59PM**

PROBLEM:

For better or for worse, the HR department now regards you as a *genius* when it comes to writing shell scripts for the company (and for providing useful utilities to help evaluate your co-workers!). And now they want you to do more for them…

It has been suspected that some employees have been "padding out" their weekly reports with lots of repetitive sentences and extraneous words to make their reports look long and impressive without having significant substance. Since you did such a good job with analyzing `git` output, they'd like you to write a utility to count occurrences of all words in a plain text file (which is how the weekly reports are submitted by all of your coworkers). For example, if the following simple text file were a report:

```
This is a test
This is only a test
This is not a real emergency
This is a silly test
This is fun
```

The HR department would like your script to produce the following report:

```
RESULTS
==================================================
'silly' occurs 1 time.
'is' occurs 5 times. WARNING!
'fun' occurs 1 time.
'this' occurs 5 times. WARNING!
'a' occurs 4 times.
'only' occurs 1 time.
'not' occurs 1 time.
'test' occurs 3 times.
'real' occurs 1 time.
'emergency' occurs 1 time.
```

Here are the requirements:

1. Name your script wordCount.sh. It will take one argument (the file to check)
2. Use a bash *associative* array (more info below) to store each word and the corresponding count of that word (word is key, count is value)

3. Read the file line by line, and for each line you should iterate over every word and either insert a new entry in the array (if the word *hasn't* been encountered before) with a value of 1 OR (if the word *has* been encountered before) increment the value by 1.
4. Print out the results of the counting exercise in a report like the one shown above:
    a. Convert the word to all lowercase so that the same word is counted regardless of capitalization throughout the input file
    b. If the word only occurs once, print out a line that says the word occurs "1 time"
    c. If the word occurs 2-4 times, print out a line that says the word occurs "n times"
        i. Where n is the actual number of times it occurs
    d. If the word occurs 5 or more times, print out a line that says the word occurs "n times" AND print out a "WARNING!". (That will alert the HR folks!)
5. The command execution should look something like this:

```
$ ./wordCount.sh simplefile
```

The command should be called `wordcount.sh` and take one parameter which is the file it is going to analyze. Your script should detect the following error conditions and print out the following messages for each (use the EXACT message below as we'll have grading scripts looking for them):

```
$ ./wordCount.sh
usage: wordCount.sh <filename>

$ ./wordCount.sh noSuchFile
wordCount.sh: noSuchFile: Not a file
```

IMPORTANT: In the above example, noSuchFile represents *a file that does not exist* and **NOT** the argument named "noSuchFile"

GETTING THE SAMPLE FILE AND ABOUT PUNCTUATION

You can test your script with the `simplefile` file provided in CMS. We'll use something a little different when grading but it will still be made up of lines of words with no punctuation. Punctuation presents a bit more of a challenge—and it is doable—but since we had a harder assignment last week I wanted to make this assignment simpler. So you may assume that there will be NO PUNCTUATION in the file and only words.

NOTES (IMPORTANT! PLEASE READ!):

PLEASE make sure you do the following when solving this problem. First, you will be storing the counts for all words in an *associative* array. These can be declared in your script with the following command (look it up if you don't trust me ☺):

```
declare -A ARRAYNAME
```

If you do not use an associative array, you will not be able to use strings as indices. What's worse is that you'll need at least bash version 4.0 and the default bash on MacOS X is version 3.x (and will not work). So you may have to use ugclinux if you don't have access to a bash 4.0 shell on your local machine.

ALSO: When printing out the report you will undoubtedly be iterating through all the indices in the associative array. This might not be clear from the lecture slides, but to iterate through the indices you will use the following for loop:

```
for index in "${!ARRAYNAME[@]}"; do
```

If you did not use the `!` in that expression, you would be iterating through all the values in the array (and not the indices).

ALSO: When looking at the final count for each individual word (and then deciding on which message to print) please **USE A `case` STATEMENT!**. One could argue that an `if-elif` would be better because we're going to be making decisions on what to print based on a numeric comparison whereas `case` matching uses pattern matching. That being said, I want you to get experience using `case`, so you will need to use it in order to get a 1 on this assignment.

FINALLY: Use a *function* inside your script to do the actual word counting. It should look something like this:

```
function countWords {
     # Your implementation here
}
```

The function should take one argument (the file passed into your script) and should probably use process substitution to allow the loop(s) that will be in that implementation to manipulate your array variable and still have its values intact when you exit the function. The function can appear early in your script file… execution will skip over it until it is explicitly called. So, you can use a format such as:

```
#!/shebang

# any initializations for the whole script

function countWords {
     # Your implementation here
}

## main script starts here
# Check for proper arguments
```

```
countWords $1
# print report
```

FILES TO SUBMIT:

1. wordCount.sh