

# **Programming Assignment #3**

## **CS253, Fall 2017**

### **Due: Tuesday, Sept. 19<sup>th</sup>**

#### ***“Punctuation and Capitalization”***

#### ***Motivation & Description***

For PA2, you wrote a program that divided an input file into whitespace-delimited strings, counted how often each string occurred, and printed out the strings and counts in lexicographic order. Sometimes, however, your PA2 program sees two words as different that “should” be the same, like *is* and *is!*. This week you will expand on PA2 to fix two of its shortcomings: punctuation and capitalization.

In PA2, the input file was divided into strings using whitespace. PA3 does the same, but will further divide the resulting strings as a result of punctuation. For example, the string *document!* will become two strings, *document* and *!*. We will keep the sequences of punctuation symbols as their own strings, but we will separate them from the words they are attached to.

Punctuation does not always occur at the end of a word. Sometimes writers forget spaces, as in the example you are reading. In this case, your PA2 program would produce *spaces,as* as a single string. PA3 should divide the string at the comma, producing three strings: *spaces*, *,* (a comma), and *as*. In general, whitespace delimited strings should be divided such that every string contains either only alphanumeric characters or only non-alphanumeric characters (although see caveat below). For example, the input “What...?!Oops!” would be divided into the string “What” followed by the string “...?!”, the string “Oops”, and finally the string “!”. Your program’s output will therefore include both words and punctuation strings, both of which will have counts.

There are always exceptions in text processing, however. In this case, there are three exceptional cases where punctuation does not break a word:

1. Apostrophes do not break words. For the purposes of this assignment, apostrophes should be treated as alphanumeric characters. For example, *can’t* and *program’s* are single words.
2. A comma is not a punctuation mark if it occurs between two digits. For example, *20,000* should be considered a single word.
3. A period does not break a string if it occurs both (1) before a digit and (2) after a digit or a space. *2.0* is a single string; so is *.01*

Another source of false differences between strings is capitalization of the first letter. In English, we capitalize words for two reasons: either they are the first word in a sentence, or they are proper nouns (for example, *English*, which is the name of a language). The problem is that for proper nouns, capitalization matters. The name *Herb* is not the same as an *herb*, nor do we know how many bills *Bill* and *Will* will pay. In general, we do not want to equate the strings *herb* and *Herb*, unless of course *Herb* was only capitalized because it was the first word in a sentence, in which case we are not sure what to do.

So here is the rule. If the first letter in a word is uncapitalized, nothing is changed (e.g. it remains uncapitalized). If the first letter of a word is capitalized and it is not the first word in a sentence, then again nothing is changed (e.g. it remains capitalized). If a word is an acronym, its capitalization is not changed. If a word contains at least one digit (e.g. *Windows10* or *iPhone8*) it remains unchanged. However, if a word is capitalized, is not an acronym, does not contain a digit, and is the first word in a sentence, then we don't know (yet) whether it should remain capitalized, so we mark it as ambiguous by prepending a + sign. For example, the word *For* at the beginning of this sentence should be marked by changing it to *+For*. In the worst case, this means that your output could contain three separate counts for *Herb*, *+Herb*, and *herb*. (Note that ambiguous capitals must be marked after dividing the strings by punctuation, otherwise the + would be stripped off the word again.)

## **Definitions**

**Acronym:** a string that contains a capital letter that is not its first letter (e.g. NATO, iPhone)

**Alphanumeric:** the characters {A,...,Z}, {a,...,z} and {0,..,9}.

**Start of sentence:** a string that is not punctuation and is either (1) the first string in the file, or (2) follows a punctuation-only string that contains a period, exclamation point, or question mark.

## **Task**

Your program will take a single file name as an argument. The text file contains strings separated by whitespace. These strings should be further divided by punctuation as described above, and ambiguously capitalized words (i.e. words that begin with capital letters that are the beginnings of a sentence) are marked with a '+' in front of the ambiguous capital letter. Your program will count how many times each unique word (string) appears, after strings have been divided into strings of punctuation characters and strings of (generalized) alphanumeric characters, and after ambiguous capitalizations have been marked with +. It should write one line of output (to `std::cout`) for every unique string that appears in the file. Each line of output should contain the string followed by a space and then the number of times the string appeared. The strings should appear in lexicographic order.

For example, if the input file contained "This is a test, a test it is!!" (quotes not included), then the output should be:

```
!! 1
+This 1
, 1
a 2
is 2
it 1
test 2
```

## **Submitting Your Work**

To submit your program, first create a single tar file containing all of your source files (i.e. your .cpp and .h files) and your makefile, but not your compiled files (no executable or .o files, please). Then submit the tar file as PA3 on Canvas.

## **Grading Your Homework**

To grade your assignment, the GTAs will unpack your archive in an empty directory. They will compile your code by typing ‘make’. This command must compile your code from scratch, and it must produce an executable called PA3 in a directory called PA3. If your code does not compile using make, you will receive a zero for the assignment. Compiling your code should not produce any warning messages.

Although we will not deduct points for warning messages, we may in future assignments so get in the habit of making sure your code compiles cleanly. Assuming your program compiles, it will be graded by how it performs on test cases, including test cases with errors in the input file. If the input file contains errors, you will receive full credit if your program returns -1 to main. (In case of an error, it should also print a meaningful error message to std::cerr.) If the input does not contain errors, your program should return 0 to main.

## **Hints**

- For PA2, I recommended that you first make a vector of strings, then sort the vector, then collect counts. For PA3, I recommend that you break up strings due to punctuation while you make the vector of strings, and then mark ambiguous capitalization by scanning down the vector of strings *after* they have been divided by punctuation but *before* you sort it.
- This assignment is not quite as simple as PA2. You have more special cases (like apostrophes and commas in numbers and acronyms). Start thinking about design. How many objects do I need. What *public* methods should they have? If your code is easy to read and understand, it will be easier to work with in PA4, when things get a lot more complicated.
- Start thinking more about test cases, too. Not just test cases that should throw errors, but complex combinations of alphanumerics and non-alphanumerics.

## **Policies**

All work you submit must be your own. You may not submit code written by a peer, a former student, or anyone else. You may not copy or buy code from the web. The department academic integrity policies apply.

You may not submit your program late. To receive credit, it must be submitted on Tuesday, Sept. 19th. There is no late period. The exception is an unforeseeable emergency, for example a medical crisis or a death in the immediate family. If an unforeseeable emergency arises, talk to the instructor.