CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

http://www.cs.arizona.edu/classes/cs127b/fall15/

# Program #3: Word Count

*Due Date: September 24th, 2015, at 9:00 p.m. MST*

**Overview:** The UNIX operating system (and its variants, of which Linux is one) includes quite a few useful utility programs. One of those is `wc`, which is short for Word Count. The purpose of `wc` is to give users an easy way to determine the size of a text file in terms of the number of lines, words, and bytes it contains. (It can do a bit more, but that's all of the functionality that we are concerned with for this assignment.) Counting lines is done by looking for "end of line" characters (`\n` (ASCII 10) for UNIX text files, or the pair `\r\n` (ASCII 13 and 10) for Windows/DOS text files). Counting words is also straight–forward: Any sequence of characters not interrupted by "whitespace" (spaces, tabs, end–of–line characters) is a word. Of course, whitespace characters *are* characters, and need to be counted as such.

A problem with `wc` is that it generates a very minimal output format. Here's an example of what `wc` produces on a Linux system when asked to count the content of a pair of files; we can do better!

```
$ wc prog3a.dat prog3b.dat
   2    6   38 prog3a.dat
  32  321 1883 prog3b.dat
  34  327 1921 total
```

**Assignment:** Write a Java program (completely documented according to the class documentation guidelines, of course) named `Prog3.java` that counts lines, words, and bytes (characters) of text files. The output format is shown in the Output section, below.

The user is to be able to supply the name(s) of the file(s) in two ways. The first is on the command line, as `wc` expects. We saw how to read command-line arguments recently, and there's an example program that demonstrates how to do it (`T01n24`). If there are no command-line arguments, your program is to display some usage information and prompt the user to enter the file name(s) on the keyboard.

**Data:** On the class web page you can find the two files, `prog3a.dat` and `prog3b.dat`, that I used to create the example above. These are just sample input files, meant to get you thinking about how `wc` behaves. You should plan to create several sample input files of your own to test further the behavior of your program. You can be sure that your section leader will be grading your program by testing it on a variety of files. The more testing you do, the greater the likelihood that your program will work correctly when graded.

**Output:** Your program is to produce counts of the number of lines, words, and characters (bytes) found in each readable file provided on the command line, and the output is to be displayed to the user in the well–structured, clearly–labeled format shown below, starting with a blank line. The five lines above the table (a blank line, the description, another blank, and the prompting line) are to be displayed only when no filenames are given on the command line. Here is an example of the output we expect when the user gives no file names on the command line but provides two when prompted:

```
This program determines the quantity of lines, words, and bytes
in a file or files that you specify.

Please enter one or more file names, comma-separated: prog3a.dat, prog3b.dat

   Lines     Words     Bytes
--------  --------  --------
       2         6        38  prog3a.dat
      32       321      1883  prog3b.dat
------------------------------------------
      34       327      1921  Totals
```

(Continued ...)

If the user supplies the name of only one existing, readable file, the last two lines (the line of hyphens and the line of totals) are not to be displayed. If, when prompted for file names, the user fails to give any usable file names, your program is to terminate after displaying some helpful instructions about what the program does, what input is expected from the user, and what output the user can expect to receive. As shown, we expect the list of file names to be comma–separated when received from the direct prompting (no commas are typed when names are given on the command line, in keeping with common UNIX command line behavior).

**Turn In:** Use the 'turnin' page to electronically submit your `Prog3.java` file to the `cs127bsXp03` directory at any time before the stated due date and time.

### Want to Learn More?

- `wc` is a standard UNIX utility program. As such, it has on–line documentation. Alas, that documentation talks a lot more about the various options than about how `wc` behaves. But, if you're curious, you can do a Google search for `wc man page` or just type `man wc` in a Linux terminal window.
- Wikipedia has a brief page on `wc` that may be helpful: `http://en.wikipedia.org/wiki/Wc_%28Unix%29`

### Hints, Reminders, and Other Requirements:

- Notes on classes you may or may not use:
  - You may **NOT** use the `Scanner` class on this assignment. We've talked about some of Java's other file classes; now's the time to learn how to use them.
  - It's entirely possible to do this assignment with just the classes we've covered in class. Java has other file I/O classes, and you're welcome to use them if you wish (but, again, not `Scanner`). **However**, you may find that, for some tasks, such classes are more trouble than they are worth. Instead, as an example, you could look into something like `String`'s `split()` for comma-separated lists (`split()` needs a very simple regex; that's OK). As another example, because your program needs to count all of the characters in the file, reading the input a character at a time and handling characters such as the newlines and commas when they are encountered is a good approach.
- Notes on counting file components:
  - To repeat from the Overview: When counting bytes of a file, you need to remember to count the end–of–line characters, too, so that you get the correct total for both UNIX and Windows/DOS style text files. Think about this when you choose the file class(es) and methods you'll be using.
  - Some text editors place a newline character (or carriage return / newline pair) at the end of the last line of a text file, and some do not. For this assignment, assume that a line must end with one of those two style of markers to be counted as a line.
  - In DOS text files, the `\r` and `\n` line terminators are separate characters, and your program should count them as such.
- This program does not lend itself to an reasonable object–oriented design. Rather than trying to dream up a way to do this with instantiable classes, you'll be better off creating just the `Prog3` class and writing some static methods for `main()` to call. Do not write a program that has only a huge `main()` method!
- If you dig into all that `wc` can do, you may wonder if you're expected to write your program to do everything that `wc` does. No! You are not expected to have your program respond to any of `wc`'s command line flags. However, it should behave like `wc` in that it is to count lines, words, and bytes from one or more files whose names are acquired from the user.
- Are you wondering why UNIX's `wc` doesn't produce a more attractive output format? There's a practical reason: Often, the output of one UNIX utility program is used as input to ("piped to") another utility program. Using a no–frills output format makes this easier to accomplish.
- As always, adequately document your program source code according to our class guidelines.
- Finally, and as always, start early. File processing is often a tricky business, good documentation takes time to write, and you'll want to ask questions early to get answers early.