

<http://www.cs.arizona.edu/classes/cs227/spring07/>

Program #5: Word Count

Due Date: February 26th, 2007, at 10:00 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 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.

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:

```
  2   6   38 prog5a.dat
 32 321 1884 prog5b.dat
 34 327 1922 total
```

We can do better!

Assignment: Write a Java program (completely documented according to the class documentation guidelines, of course) that counts lines, words, and bytes of text files, and produces the output in the format shown below:

```
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: prog2a.dat, prog2b.dat
```

```
  Lines      Words      Bytes
-----
     2         6        38 prog5a.dat
    32       321     1884 prog5b.dat
-----
    34       327     1922 Totals
```

Note that if the user supplies only one filename, the last two lines (the separation line and the totals) are not to be displayed. If the user fails to give any 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.

(Continued ...)

Data: Assuming that you mapped your X: drive as suggested in Program #4, in it you should find a folder named **Source**. In that folder should be two files creatively named `prog5a.dat` and `prog5b.dat`. I used these files to create the examples shown above. They are also linked to the class web page. 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: As shown above, your program is to produce counts of the number of lines, words, and characters (bytes) found in each file provided on the command line, and the output is to be displayed to the user in the well-structured, clearly-labeled format used for the above example. When multiple filenames are provided, a set of totals is to be displayed. If no filenames are given, usage information will be displayed.

Turn In: Electronically submit your `Prog05.java` file using the usual

[section leader first name]_S[your section #]_P[program #]_[your last name]_[your first name]
naming convention.

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`.

Hints, Reminders, and Other Requirements:

- Notes on classes you may or may not use:
 - You may **NOT** use `Scanner` for file or keyboard I/O on this assignment. We've talked about some of Java's other file classes; now's the time to learn how to use them.
 - You **MAY** use classes such as `StringTokenizer` in this assignment, but you may find that for some tasks such classes are more trouble than they are worth. (See pages 707-8 in Koffman for a `StringTokenizer` example.)
- Notes on counting file components:
 - 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) you'll be using.
 - As we saw in class, 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 kinds of markers to be counted as a line.
- This program does not lend itself to an obvious 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 `Prog05` 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 better output format? There's a pretty good 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.
- Be sure that you have adequately documented your program source code according to the class programming style guidelines.
- Finally, and as always, start early. File processing is often a tricky business.