## CPSC 229, Fall 2008 — Computer Assignment #2

*This assignment is due on Monday, November 3. Turn in answers in writing for problems 1 and 2. Place the file that you create in problem 2 and a copy of the program that you write for problem 3 into the directory /classes/f08/cs229/zz9999, where you should replace "zz9999" with your own username. I will compile, run, and print out the program myself. You do not have to turn in a printout.*

**1.** (**Fun with *grep*.**) The *egrep* command was covered in the regular expressions handout. The handout also described how to use "pipes" on the command line to apply a sequence of commands. For this exercise, you will use *egrep* and other commands to answer some questions about some files that can be found in the directory */classes/f08/cs229*. You can *cd* into that directory to run the commands; there is no need to copy the files into your own directory (especially since one of them is 84 megabytes!).

**a)** The file *www_access_log.txt* contains a list of requests made from the Internet to the web server on math.hws.edu last week. Some of these requests are for files in student user accounts. A typical request for such a file starts with `"GET /~zz9999/`, where *zz9999* is replaced with the actual username of the student. The student user name can be any two letters followed by any four digits. For this part of the exercise, you should find out how many of the 380877 lines in the file *www_access_log.txt* are requests for files in student accounts. You can use an *egrep* command to extract lines. To count the lines, you can pipe the output of the *egrep* command into the command "`wc -l`" which counts the number of lines in its input. Report the command that you used and the number that you got for an answer.

**b)** Some of the lines in *www_access_log.txt* come from search engines such as Google. These can be recognized because they match the regular expression "`&q=[^&]+&`". Lines generated by Google, as opposed to other search engines, also contain the string "`google`". Find out how many lines in *www_access_log.txt* were generated by google searches. (Note: You can pipe the output from one *egrep* command into another `egrep` command.) Report the number of lines and the full command line that you used to get your answer.

**c)** As explained in the handout, the `-o` option on an *egrep* command causes it to print out all matching substrings, instead of printing out the lines that contain those substrings. You can use this to get a listing of all the "words" in a file, where we can define a word to be any sequence of letters. The command "`tr A-Z a-z`" will convert all upper case letters in its input into lower case letters; other than that, the output is the same as the input. Use an *egrep* command, along with some pipes, to find out how many words there are in the file *regular_expressions.html*. Once you've done that, add a "`sort -u`" command to the pipe to find out how many different words there are in the file, after all the words are converted to lower case. Report both of the answers and the command lines that you used to get them.

**2.** (**Editing Text with Regular Expressions.**) The file *passwd.txt* in */classes/f08/cs229* contains a list of 365 user accounts on the Math/CS network, in the format used by the standard UNIX "passwd" file. Each line of the file contains seven fields, separated by colons (:). The first field in the username. The fourth field in the real-world name. I would like you to edit the file so that each line contains **only** a user name and a real-world name. Use a text editor, and use regular expression find-and-replace to do the editing in one step. Make sure that you've read the section on "Backreferences" in the handout. Section 3.3 in the textbook might also help. You can use the Replace command in either *nedit* or *kate*; other text editors might also work. It will probably take several tries for you to get the regular expression and replacement text correct. Note that after doing a global find-and-replace, you can use the Undo command to get back to the previous text (or, you could just open the file again).

**3.** (**Regular Expressions in Java.**) For this exercise, you should write a program to evaluate simple arithmetic operations typed in by the user, but you should do it using a regular expression. Legal inputs will consist of two positive integers separated by one of the operators `+`, `-`, `*`, `/`, or `^`, where `x ^ y`, means $x^y$ and can be evaluated using `Math.pow(x,y)`. There can be spaces next to the operator, but these spaces are not required.

Your program should accept more than one line of input and should end when the user enters an empty line. If the user enters illegal input, the program should not crash; it should output an error message and continue with another line of input. If there is no error, then the program should print the value of the user's expression. You don't have to add comments to your program, but you are required to use good programming style aside from that, and you should provide a decent user interface (with reasonably labeled input and output).

Your program must use a regular expression to match the user's input and extract the three pieces of information that you need (the two numbers and the operator). You will need to use the *Pattern* and *Matcher* classes that are discussed in the handout. To extract the needed information, you will need the *group(n)* method from the *Matcher* class.

(You will also need to be able to convert a *String* into an *int*. Recall that this can be done with `Integer.parseInt(str)`, where *str* is the string. This method throws a *NumberFormatException* if the string does not represent a legal integer.)