

This assignment is due on Friday, November 5. You should be able to complete most of Part 1 during class on October 27. For Part 1, turn in written (or typed) answers. For Part 2, place copies of your programs in your homework folder in /classes/f10/cs229/homework.

Part 1: Lab

The *grep* and *egrep* commands were covered in the regular expressions handout. The handout also described how to use “pipes” on the command line to apply a sequence of commands. One useful command is *wc -l* which counts the number of lines of input (from the previous command in the pipe). Also useful is *sort*, especially *sort -u*. You can pipe into *less* or *head* if you want to view the output.

For this lab, you will use *egrep* and other commands to answer some questions about two files that can be found in the directory */classes/f10/cs229*. You should *cd* into that directory to run the commands; **do not** copy the files into your own directory (especially since one of them is extremely large!).

The file *yppasswd* contains a list of user accounts on the math/cs network, in the format commonly used for passwords. A line in the file contains 7 fields separated by colons (:). The first field is the user name, the second is always an “x” (in our file), while the third and fourth are integers. The fifth field is the real name of the user. Look up your entry by doing *grep zz9999 yppasswd*, replacing *zz9999* with your own user ID.

The file *access.log* contains a list of requests sent from web browsers to the web server on math.hws.edu in a 10-day period last month. Each line of the file starts with the IP address of the computer that sent the request. The IP address consists of 4 integers, separated by periods. A little later is the request itself, enclosed in double quotes. There are several possible types of request, but we are interested in requests that start with *GET*; these are simple requests for web pages. In such a request, *GET* is followed by a space, then the path to the file that is being requested, then another space. (Take a look at the first few lines in the file!)

1. How many student accounts are there in the *yppasswd* file? A student account is one that has a user name that matches the regular expression `[a-zA-Z]{2}[0-9]{4}`. Use *egrep* and *wc -l* to find out. Report the number of student accounts **and** the command that you used to determine the number.
2. IP address on the HWS network begin with 172.30., 172.20., or 172.25. How many of the requests in the *access.log* file came from computers on campus? Use *egrep* and *wc -l* to find out. (Use “^” in your regular expression to make sure that you are looking at an IP address at the start of the line.) Use an “*egrep -o*” to obtain a list of just the IP addresses, and add a *sort -u* to your pipe to determine how many **different** IP addresses those requests came from. Report the two numbers that you found and the commands that you used to get them.
3. How many different files from student accounts were requested from off-campus computers? A request for a file in a student account starts with

`"GET /~zz9999`

and ends with the next space. (The “*zz9999*” can be any student ID.) You will need a pipe containing several commands to answer this question. Warning: When you match the file name, make sure that you don’t include any spaces. You will need a pipe containing several *egrep*

commands to extract the file names. Build up your pipe one command at a time, testing that it works. (Piping into *less* or *head* is useful for testing.) Report the command that you used and the number that you got for the answer.

- I would like a list of email addresses of students with accounts in the `yppasswd` file. The email address should be formatted exactly as in this example:

```
"Doe, John" <zz9999@hws.edu>
```

Refer to the regular expressions handout for the section on “Perl Substitutions with *perl -pe*”. Write a perl substitution command that will convert a line from the `yppasswd` file into an email address. Use your command to create the email list that I asked for. (Combine it with an *egrep* command that extracts the student accounts from the file.) Report the perl command that you used.

Part 2: Programming

- 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. Use a single *Pattern*, and create a *Matcher* for each line of input. 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. Alternatively, you could consider using the *BigInteger* class to avoid problems with integer overflow.)

- Write a Java program that will take an HTML file as input and will output a list of all the pages to which the given HTML file links. Examples of links in an HTML file include:

```
<a href="http://google.com">  
<A target="mainframe" href='data321.html'>  
<a id='link23' HREF = "file23.html" target="_TOP">
```

The link must start with `<a` and contain `href=`. (These are case-insensitive.) There can be spaces around the `=`, but not after the `<`. The page is in single or double quotes after `HREF=`. Your program should use a *Pattern* that will recognize such links and will extract the page as a group. Use a *Matcher* for each line of input, to test whether it contains a link to a page and, if so, to extract the page.