This homework is due in class on Friday, November 7. The homework consists of a few questions on regular expressions as we have been covering them in class, plus a computer/programming lab that you will start in class on Friday, October 26 and finish for homework.

*You will not be tested on regular expressions as they are used on the computer—you will only encounter those on this homework. Note that aside from the computer work, your regular expressions should use only the special symbols +, *, ·, (,), ε, and Φ, as covered in Section 3.2. (However, I will accept | in place of +.)*

1. Give an English language description of the language generated by each of the following regular expressions over the alphabet $\{a, b\}$.

a) $(a + b)^*bbb(a + b)^*$ **b)** $a^*ba^*ba^*ba^*$ **c)** $a^*(b + \varepsilon)a^*(b + \varepsilon)a^*(b + \varepsilon)a^*$
d) $a^*ba^*ba^*b(a + b)^*$ **e)** $b^*a(b^*ab^*ab^*)^*$

2. Write a regular expression that generates each of the following languages over the alphabet $\Sigma = \{0, 1\}$. In each case, explain how your answer works.

a) $\{x \in \Sigma^* \mid x$ begins with a 0 and ends with a 1 $\}$

b) $\{x \in \Sigma^* \mid |x| > 3\}$

c) $\{x \in \Sigma^* \mid |x|$ is an even integer $\}$

d) $\{x \in \Sigma^* \mid x$ contains at least one of the substrings 000 or 111 $\}$

e) $\{x \in \Sigma^* \mid x$ contains both of the substrings 000 and 111 $\}$

Computer Lab

The remaining questions are to be started in class on Friday, October 26. For the information that you need, you can refer to the handout on Regular Expressions on the Computer. If you need information that you can't find there, ask.

3. The file */classes/f07/cs229/access.log* is a log file for the web server that runs on math.hws.edu. It contains information about requests sent to the server over the course of a few recent days. Using the command line, cd into the directory */classes/f07/cs229* to make it easy to work with this file. Please do not copy it; it's 44 megabytes.

The command *wc -l access.log* can be used to find the number of lines in the file. Use *grep* to search for lines that contain the string google.com and pipe the result into *wc -l* to find out how many lines contain that string:

```
egrep 'google\.com' access.log | wc -l
```

To see the lines instead of just counting them, you can pipe the results into *less*:

```
egrep 'google\.com' access.log | less
```

Many of the web page accesses come from people who found the page by searching on Google. In that case, the entry in the access log contains the "query string" that the user typed into Google's search form. The query string begins with `q=` and it is terminated either by the ampersand character (`&`) or by the double-quote character (`"`). It cannot contain either of these

two characters. Write a regular expression that can be used with *egrep* to match such a query string. Pipe the output from *egrep 'google.com'* to another *egrep* command that will output **only** the query strings. Then add some more commands to find out how many query strings contain the word "java" (ignoring case).

Report the regular expression that you used to match query strings and the number of query strings that contained the word "java".

4. This exercise uses regular expression search-and-replace in a text editor. You will need a file to work with. I want you to have a file that contains information about student users of the Math/CS computer systems. The command *ypcat passwd* outputs information about all users. Student users have a user name that consists of two letters and four digits. To filter out the other users and store the results in a file named `student_users.txt`, use the command:

```
ypcat passwd | egrep '^[a-zA-Z]{2,2}[0-9]{4,4}' >> student_users.txt
```

However, what I really want is a file that contains legal email addresses in a format like: `"David J. Eck" <eck@hws.edu>`. Open the file in the *kate* text editor with the command *kate student_users.txt*. User kate's regular expression replace, with "placeholders", to change each line in the file into an email address in the given format. Use a regular expression that matches an entire line, with parenthesized subexpressions that match the user name and the real name. Use backreferences in the replacement text to represent the user name and real name. The replacement will be applied to the entire file, which in the end should contain lines such as: `"Christopher Hagar" <ch0170@hws.edu>`. You will probably make some mistakes along the way; Control-Z can be used to undo a replace operation.

In your report, explain the meaning of the pattern `'^[a-zA-Z]{2,2}[0-9]{4,4}'` in the *egrep* command given above. Give the regular expression and the replacement text that you used to do the search-and-replace in *kate*. (If possible, show me the result of making the change during lab.)

5. For this problem, you will write a Java program that solves a problem adapted from a student programming contest. The problem is to write a program that evaluates complex number arithmetic operations. A complex number is a pair of real numbers, separated by a comma and enclosed in parentheses. A line of input will consist of two complex numbers in this format, separated by one of the operators $+$, $-$, $*$ or $/$. Whitespace should be allowed on either side of the operator (at least). In your program, create a *Pattern* from a regular expression that describes such inputs. (Don't forget to use, for example, `\\(` to represent an actual left parenthesis in the input.) Use parenthesized sub-expressions to capture the four real numbers and the operator in the input. For each line of input, use the *Pattern* to create a *Matcher* to match the input. Use the *Matcher's* `group(n)` method to recover the real numbers and the operator from the input. You can use `Double.parseDouble(s)` to try to convert the real numbers from string form into values of type *double*. Then compute and print the value of the expression, following the rules:

```
(a,b) + (c,d) = (a+b,c+d)
(a,b) - (c,d) = (a-b,c-d)
(a,b) * (c,d) = (a*c-b*d,a*d+b*c)
(a,b) / (c,d) = ((a*c+b*d)/(c*c+d*d),(b*c-a*d)/(c*c+d*d)
```

If there is any error in the input, report it to the user. Continue processing user input until the user enters an empty line. If you want to use TextIO.java to read the input, you can find a copy in the directory */classes/f07/cp229*.

Turn in a printout of your program. If I have any doubt about whether it works correctly, I will ask you to send me a copy so that I can try it out.