CSE 391, Autumn 2019
Homework 3: Regular Expressions
Due Tuesday, October 22, 2019, 1:00PM

Special thanks to CSE 142 for the inspiration for task2

This assignment focuses on using regular expressions and related commands such as `sed` and `egrep`. The second task will focus on searching for particular sequences of DNA. A set of files you will need for this assignment are available in the file `hw3.zip`, found on the Homework page.

Submit your answers to Gradescope. In response to each question, **write the command that will perform the task described**, *not the output that the command produces*. Please be sure to write the **entire command** (including the command name and the input file) not just the regular expression required for the problem.

Task 1: Basic Navigation in a Text Editor

The following are exercises and questions are meant to help you become more comfortable with a text editor that is built into the command line. You can choose either vim or emacs. While the answers to the questions themselves are relatively easy to find by simply looking them up, **the real learning will come from you actually practicing these commands yourself**. While we won't be able to know whether you've really been practicing, this is not for our benefit, it's for yours. We also recommend getting even more practice by writing the answers to your task1.sh, task2.sh and task3.sh files using this editor ☺

For these questions, you will be using the text editor of your choice to practice searching for text using regular expressions (yes, you can use regular expressions to search for text in a text editor!) from the file `fruits.txt`.

1. What is the command to find all fruits that end in "erry"?
2. What is the command to find all fruits that contain the string "oo" *within* the word? That is, words where there are characters on either side of "oo." Note: vim requires you to escape some regular expression special characters.

Task 2: Bash Shell Commands with `egrep` to process genetic data:

For each item below, **determine a single `bash` shell statement that will perform the operation(s) requested**. Each solution must be a one-line shell statement, but you may use input/output redirection operators such as >, <, and `|`.

For these problems, you will be processing DNA data from the file `dna.txt`. Data is printed in the file in pairs of lines. The first line in the pair is the name of the DNA sequence and the second line is the DNA sequence itself. The following provides you with some context for the task, but an understanding of DNA is not required for this assignment.

DNA consists of long chains of chemical compounds called nucleotides. Four nucleotides are present in DNA: Adenine (**A**), Cytosine (**C**), Guanine (**G**), and Thymine (**T**). Certain regions of the DNA are called genes. Most genes encode instructions for building proteins (they're called "protein-coding" genes). These proteins are responsible for carrying out most of the life processes of the organism. Nucleotides in a gene are organized into codons. Codons are groups of three nucleotides and are written as the first letters of their nucleotides (e.g., TAC or GGA). Each codon uniquely encodes a single amino acid, a building block of proteins.

For these problems you will be identifying protein-encoding genes as well as other attributes of the genetic data in the file. Note that all matches for these problems will be case-insensitive. Write a command that uses `grep -E` or `egrep` to process the data in the file. Use regular expressions as appropriate. Write your commands on the indicated lines in the `task2.sh` file in the `hw3` folder.

3. Print all of the DNA sequences in the file (all of the non-names).
4. Print all of the names of DNA sequences in the file. *Hint: This will require use of a grep option, in addition to regular expressions.*

5. Print all of the valid genes in the file. That is, all of the DNA sequences that can be divided into codons. (i.e. A sequence of A, C, G and Ts and has a length that is a multiple of 3).

6. Protein-coding genes are genes that begin with ATG, end with one of TAA, TAG, or TGA, and are at least 5 codons long. Print all protein-encoding genes in the file.

7. Print the full DNA sequences that contain the word “CAT”, preceded by the name of the sequence. *Hint: man grep to look for options that alter what gets printed for a match.*

8. How many codons match “CAT”? Because codons are groups of three nucleotides, the positions of the nucleotides matter. For example, the sequence CATATG contains the codon CAT, while the sequence ACATTG does not because it is made of up the codons ACA and TTG. *Hint: You will have to process the data in two stages, first to find all of the codons, then to match codons against CAT. You may find the –o option helpful here.*

9. Print all of the DNA sequences that start and end with the same codon. (For this case, do not worry about matching start and end codons that don’t have the same casing as each other.)

Hints: Recall the **regular expression syntax** shown in class, such as a `.` for any character, `|` for "or," `[]` for character classes, `{}`, `*`, and `+` for quantities, `^` and `$` for specifying the start/end of a line, and `\<` and `\>` for specifying the start/end of a word. Also note that `grep` can use "back-references" to refer to sub-patterns captured previously between `(` and `)`, such as `\1` for the first captured sub-pattern, `\2` for the second, etc.

Task 3: More Bash Shell Commands with `egrep`:

For each item below, **determine a single `bash` shell statement that will perform the operation(s) requested**. Write a command that uses `grep -E` or `egrep`. Use regular expressions as appropriate. Write your commands on the indicated lines in the `task3.sh` file in the `hw3` folder.

10. Output all of the lines in `vars.txt` that contain valid Java variable names. Recall that a variable name must have the following properties:
    a. Must start with an underscore, uppercase letter, lowercase letter, or a dollar sign (`$`).
    b. Followed, optionally, by any combination of numbers, underscores, uppercase or lowercase letters, and dollar signs.

    For the sake of this problem you may assume that there is only one valid variable per line.

11. Output all of the method headers in the file `PointProgram.java`. A method header will have the following structure, in order:
    a. Zero or more leading spaces
    b. Either `public` or `private`
    c. Optionally, `static`
    d. A return type
    e. A method name
    f. Parentheses containing zero or more parameters, where parameters are a comma separated list of parameter type and parameter name pairs.
    g. An opening curly brace.

    You do not need to validate that the method headers form valid java code (e.g. that the java types are valid or the parameter names do not repeat etc.). You may assume that types, method names and parameters names are formed of letters only. Note that this will not capture `public static void main(String[] args) {` because of the array type `[]`, nor would it capture types with generics because of the <>. This is okay. As long as you capture all of the method headers in the given java program, other than main, consider yourself done.