Assignment 3

This assignment consists of two exercises in writing relatively simple shell scripts. The objectives when writing any script are:

- **Clarity**: The script should be easy to understand by someone with a basic knowledge of UNIX;
- **Efficiency**: The script should use the least resources possible;
- **Simplicity**: The script should be as simple as possible.

An example will demonstrate. Suppose we needed a script that would count the number of lines in a file named `molecule` containing the word 'ATOM' anywhere on the line. The following script would achieve this:

```bash
#!/bin/bash
grep 'ATOM ' molecule >| atomcount
wc -l atomcount >| answer
rm atomcount
cat answer
rm answer
```

But it is very inefficient (because it needlessly creates files and then removes them), it is hard to understand because the reader spends more time reading it and may not be familiar with certain operators such as `|`, and it is not as simple as it could be. A simple, well-documented, and efficient solution is:

```bash
#!/bin/bash
# Displays how many lines in file molecule contain ATOM as a complete word
# Written by Stewart Weiss
grep -c 'ATOM ' molecule
```

Your task is to apply these ideas as you create solutions to the following two exercises.

1. In this exercise, you are to work once more with PDB files that represent proteins. A protein consists of a sequence of **chains** of amino acids, called **residues**. Sometimes a protein has a single chain, and sometimes it has many chains. The ATOM records in the file indicate, for each atom, which chain it belongs to. Each chain is named by a one-letter code following the space after the residue name. For example, the following line:

   `ATOM 62 CE3 TRP A 7 34.108 15.499 -6.564 1.00 11.05 C`

   indicates that the atom is part of a tryptophan residue on chain A, and this line:

   `ATOM 930 N LYS B 22 38.096 60.058 67.587 1.00 27.57 N`

   indicates that the atom is part of a lysine residue on chain B.

   Your job is to write a script named `countAtoms` that expects **three** arguments. The first argument is the three letter code of an amino acid, in uppercase, such as LEU or ASN. The second is a one-letter chain name, such as A, B, or C. The third is the name of the PDB file in which to look. The script should count the number of atoms in the file that belong to the given amino acid and are in the given chain in the given file and display the number (and **nothing but the number**) on the screen (i.e., on what we call standard output). If it finds no atoms of that amino acid type in that chain, it should output the number 0. When run it would look like this:
showing that there are 380 atoms in the leucine residues in chain A and 760 in the leucine residues in chain B. Your script must check that it is given three arguments and exit with a usage message if it is not used properly. It does not have to check that the residue name or the chain name is valid, or even if the file can be opened.

2. A DNA string (also called a DNA strand) is a sequence of the letters a, c, g, and t in any order. For example, aacgttggtaaccagaactgt is a DNA string of length 21. Each sequence of three consecutive letters is called a codon. For example, in the preceding string, the codons are aac, gtt, tgt, aac, cag, aac, and tgt. If we ignored the first letter and started listing them with the second a, the codons would be acg, ttt, gta, acc, aac, aga, and act. For simplicity, we will assume that we always start reading the codons at the first letter of the string. A DNA string can be hundreds of thousands of codons long, even millions of codons long, which means that it is infeasible to count them by hand. It would be useful to have a simple script that could count the number of occurrences of a specific codon in such a string. For instance, for the example string above such a script would tell us that aac occurs three times and tgt occurs twice.

Your job is to write a script named `countcodons` that expects two arguments on the command line. The first argument is a three letter codon string such as aac or tgt. The second is the pathname of a file containing a valid DNA string with no newline characters or white space characters of any kind within it; it is just a sequence of the letters a, c, g, and t and nothing else. If it is given two valid arguments, the script will output a single number, which is the number of occurrences of the given codon in the given file. As with the first exercise, it should output absolutely nothing but that number. If it finds no occurrences, it should output 0. For example, if the above string is in a file named dnafile, then it should work like this:

```
$ countcodons ttt dnafile
1
$ countcodons aac dnafile
3
$ countcodons ccc dnafile
0
```

**Warning:** if given valid arguments, the script is not to output anything but a number. It will lose points if it does. The script should check that it has two arguments and exit with a usage message if it does not. It is not required to check that the file is in the proper form, or that the string is actually a codon. However, for 10% extra credit, it should print an error message and exit if the second argument cannot be opened or is not a file containing only the four letters, a, c, g, and t.

**Hint:** You will not be able to solve this problem using the `grep` command alone. There are a number of commands that might be useful, such as `sort`, `cut`, `fold`, and `uniq`. One of these makes it very easy. Find the one.

**Submitting the Solution**

Create a directory in `/data/bios/b/student.accounts/cs132/projects/project3` named `hwk3_username`. Make sure that you use only lowercase letters in the directory name so that it matches your username exactly. Remember, this is a directory, not a file. Give your directory permission `rwx------` (700). Put your two scripts into this directory, with the names specified above, e.g., `countatoms` and `countcodons`. These scripts should have permission `rwx-xr-x`. (No one can see them since they are protected by the permissions on the directory.) Make sure that each script has your name in a comment line at the top.