Assignment 4

1 Overview and Background

Many of the assignments in this course will introduce you to topics in computational biology. You do not need to know anything about biology to do these assignments other than what is contained in the description itself. The objective of each assignment is for you to acquire certain particular skills or knowledge, and the choice of topic is independent of that objective. Sometimes the topics will be related to computational problems in biology, chemistry, or physics, and sometimes not.

This particular assignment is an exercise in extracting information from files that are too big for mere mortals to process manually. The real power of computers is that they can do simple things extremely quickly, meaning millions, perhaps billions of times per second, much faster than people can. There is a kind of file called a *PDB* file that contains structural information about proteins, nucleic acids, and other *macromolecules*. A macromolecule is just a big molecule. *Macro* means big. *PDB* is an acronym for the *Protein Data Bank*. PDB files can be downloaded from the Protein Data Bank at http://www.rcsb.org/pdb/home/home.do.

A PDB file contains information obtained experimentally, usually by either *X-ray crystallography*, *NMR spectroscopy*, or *cryo-electron microscopy* [1]. (You do not need to know this to do the assignment, but it is important for those who intend to pursue a bioinformatics concentration.) These files completely characterize the molecule, providing, for example,

- the three-dimensional positions of every single atom in the file,
- where the bonds are,
- which amino acids it contains if it is a protein[2] (or nucleotides if DNA or RNA) ,

and much more. The information is not necessarily exact. Associated with some of this information are confidence values that indicate how accurate it is. A PDB file is a plain text file; you can view its contents in any text editor, such as *gedit* or *nedit*, or with commands such as `cat`, `more`, and `less`. Each line in a PDB file begins with a word that characterizes what type of line it is. These individual lines are called *records*. For example, some lines start with the word `REMARK`, which means they are comments about the file itself, or about the experimemt through which the data was collected. Some lines start with `SOURCE`, and they have information about the source of the data in the file. Some lines start with words such as `MODEL`, `CONECT`, `ATOM`, and `HETATM`. Each has a different meaning in the file. Take a look at some of the PDB files in the directory `/data/biocs/b/student.accounts/cs132/data/pdb_files` before you read any further, so that you can see what they contain. I suggest picking files that are small, meaning smaller than a megabyte in size.

Proteins are chains of amino acids. Amino acids are organic compounds that carry out many important bodily functions, such as giving cells their structure. They are also instrumental in the transport and the storage of nutrients, and in the functioning of organs, glands, tendons and arteries. Amino acids have names such as *alanine*, *glycine*, *tyrosine*, and *tryptophan*. They are also known more succinctly by unique three-letter codes. The table below lists the twenty standard amino acids with their three-letter codes.

[1] For a summary of what these methods are, see http://www.pdb.org/pdb/static.do?p=education_discussion/Looking-at-Structures/methods.html

[2] Amino acids are the building blocks of proteins, which may contain many thousands of them.

Amino Acid Name	Code
Alanine	Ala
Arginine	Arg
Asparagine	Asn
Aspartate	Asp
Cysteine	Cys
Glutamate	Glu
Glutamine	Gln
Glycine	Gly
Histidine	His
Isoleucine	Ile
Leucine	Leu
Lysine	Lys
Methionine	Met
Phenylalanine	Phe
Proline	Pro
Serine	Ser
Threonine	Thr
Tryptophan	Trp
Tyrosine	Tyr
Valine	Val

Each line that starts with the word `ATOM` represents a unique atom in the protein. So do the lines that start with `HETATM`, but these are atoms in water molecules surrounding the particular protein when it was crystallized, and we want to ignore them for now. Lines that start with `ATOM` contain the three-letter code for the amino acid of which that atom is a part. For example, an atom line for an atom in a phenylalanine molecule looks like this:

```
ATOM 3814 N PHE J 24 -17.763 -7.816 -12.014 1.00 0.00 N
```

The three-letter code is always in uppercase. The exact form of a PDB file is standardized. The standard is revised every few years. The most recent standard that describes the format can be found at:

https://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html

On that page you can scroll down to the `Coordinate Section` and find the link to the `ATOM` record format. There you will see that the line for an atom is defined by the following table:

COLUMNS	DATA	TYPE	FIELD DEFINITION
1 - 6	Record name	ATOM	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name.
17	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name.
22	Character	chainID	Chain identifier.
23 - 26	Integer	resSeq	Residue sequence number.
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms.
47 - 54	Real(8.3)	z	Orthogonal coordinates for Z in Angstroms.
55 - 60	Real(6.2)	occupancy	Occupancy.
61 - 66	Real(6.2)	tempFactor	Temperature factor.
77 - 78	LString(2)	element	Element symbol, right-justified.
79 - 80	LString(2)	charge	Charge on the atom.

Notice that the fields are not necessarily separated by white space or by special characters. Instead they are defined by their exact column numbers on the line. For example, the amino acid name is in columns 18 through 20. (The table calls it the ***residue name***.) At this point it would be beneficial to you to make sure that the font you use to view the files is a fixed-width font, so that you can see where the columns are. By default, the terminal in a typical Linux installation uses a *monospace* (fixed-width) font, and so viewing the files using `cat`, `more`, `less`, or any terminal-based editor will be fine.

Suppose for a moment that you had access to several PDB files representing various proteins and you needed to know how many atoms within that protein belonged to a particular type of amino acid, i.e., how much of that protein was made up of a given amino acid, not by weight but by atom count. You could open the file and start counting the appropriate lines by hand. This would take a very, very, long time. Instead, you could use your knowledge of the tools available in UNIX to solve the problem in a few minutes. There are commands in UNIX that you have learned about in this class so far that you can use to determine such things as how many atoms are in a PDB file, how many atoms of a specific type are in the file, or more complex information such as how many atoms of a given amino acid are at a certain distance from a given point in the protein. These commands are relatively easy to use, assuming you have a little ingenuity. You will have to read the man pages for them.

2 Tasks

This assignment consists of two tasks, one related to PDB files and the other related to nucleotide strings. The files that you can use for input for the first task are in the directory

```
/data/biocs/b/student.accounts/cs132/data/pdb_files.
```

Those for the second are in

```
/data/biocs/b/student.accounts/cs132/data/dna_textfiles.
```

1. Write a script named `atomcoordinates` that will expect on the command line two arguments: a valid three-letter code of a standard amino acid in uppercase, and the name of a PDB file. Given a valid amino acid name and a PDB file that it can open, the script will display, for each record of that type that it finds in the file, a line of output containing the atom's *serial number*, its *three-letter amino acid name*, and its x, y, and z *coordinates*. For example, a line in the PDB file that looks like this:

   ```
   ATOM 18 CB GLN A 3 83.556 52.126 45.080 1.00 26.06 C
   ```

   would cause the following output line to be displayed:

   ```
   18 GLN 83.556 52.126 45.080
   ```

   because the atom's serial number is 18, its amino acid name is `GLN`, and its coordinates are 83.556, 52.126, and 45.080. These five pieces of information must be separated by white-space such as blanks. Your script has to extract the serial number, amino acid name, and the coordinates from the line and display them. Your job is to decide which filters, or combination of filters, can achieve this. This will take some creativity. A hint is that all of the filters you need have been covered either during class lectures or are listed in the slides in Lesson 9. Figure out which might be useful, or even better, which are the most efficient for you to use.

   Error checking: Your script must check that it has both command line arguments, and that the file can be read. It must display a message if either of these is not true. It does not need to check whether the amino acid designation is valid, because if it is not, it should simply display nothing as its output.

2. A *DNA string* is a sequence of the letters `a`, `c`, `g`, and `t` in any order, whose length is a multiple of 3. For example, `aacgtttgtaaccagaactgt` is a DNA string of length 21. Each letter is called a *base*, and a 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`. A DNA string can be hundreds of thousands of codons long, even millions of codons long, so it is hard to count them by hand. It would be useful to have a simple utility script that could count the number of occurrences of a specific codon in such a string. For instance, in the example string above, `aac` occurs three times and `tgt` occurs twice. For simplicity, we always assume that we look for codons at positions that are multiples of three in the file, i.e., starting at positions 0, 3, 6, 9, 12, and so on.

   Write a script named `countcodon` that is given two arguments on the command line. The first is a lowercase three letter codon string such as `aaa` or `cgt`. The second is the name of a file containing a DNA string with no newline characters or white space characters of any kind except at the end after the sequence of bases; it is just a sequence of the letters `a`, `c`, `g`, and `t`. The script will output a single number, which is the number of occurrences of the given codon in the given file. ***It should output 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:

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

   The script should check that it has two arguments and exit with a usage message if it does not. It should make sure that it can open the file for reading and print a usage statement if it cannot. It does not have to check that the string is actually a codon, but it should check that the file contains nothing but the bases and possible terminating newline character.

   Hint: You will not be able to solve this problem using `grep` 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 right one.

3 Rubric

This homework is graded on a 100 point scale. Each script is worth the same number of points. Each script will be graded primarily on its correctness. This means that it does exactly what the assignment states it must do, in detail. Correctness is worth 70% of the grade. Then it is graded on its clarity, simplicity, and efficiency, as described above. Good comments are worth 15%; good design another 5%, and simplicity and efficiency the remaining 5%. Naming all files and directories correctly is 5%.

4 Submitting the Solution

This assignment is due by the end of the day (i.e. 11:59PM, EST) on Thursday, October 15. (I give a grace period of six hours after that, so it is okay to submit it by 6:00 AM of the following day.)

There is a directory in the CSci Department network whose full path name is

`/data/biocs/b/student.accounts/cs132/hwks/hwk4`.

You must put it in that directory. To submit your project, you must follow the instructions below exactly! Do not deviate from these instructions.

To be precise:

1. Login using `ssh` to `eniac.cs.hunter.cuny.edu` with your valid username and password, and then `ssh` into any `cslab` host. Do not forget this step. You will not be able to run the `submithwk` command on `eniac`.

2. If you did not do the work on one of the computers in our network, then upload the two files into your home directory. Create a directory named `your-username-hwk4` and put the two scripts into it.

3. Run the command

   ```
   zip -r your-username-hwk4.zip your-username-hwk4
   ```

   This will create the file `your-username-hwk4.zip.` The `zip` command is a special command that compresses the files in the directory and creates a new file that can later be extracted by the `unzip` command. So it will create a "zip file" named `your-username-hwk4.zip` containing your `your-username-hwk4` directory and the three files it contains. For example, I would run

   ```
   zip -r sweiss-hwk4.zip sweiss-hwk4
   ```

4. Run the command

   ```
   /data/biocs/b/student.accounts/cs132/bin/submithwk 4  your-username-hwk4.zip
   ```

   Do exactly this. Do not mistype it. The command will create a copy of the file `your-username-hwk4.zip` in the directory

   `/data/biocs/b/student.accounts/cs132/hwks/hwk4`

   It will be named `hwk4_`*`username`*, where *`username`* is your username on the network. You will not be able to read this file, nor will anyone else except for me. If you decide to make any changes and resubmit, just do all the steps again and it will replace the old file with the new one. I will be able to unzip the file, extracting whatever files you created. Do not try to put your file into this directory in any other way - you will be unable to do this.

Although these instructions may seem complicated, they simplify the way you submit your work and the way I can retrieve it. If you make mistakes, just start over. If things don't seem to work out, post a question on Piazza with the details included.