Project 5: Extracting Data from PDB Files

Overview

In this assignment, your Perl program will let a user get information about a macromolecule represented by a PDB file whose pathname is specified on the command line. The program will open and read the file, after which it will enter an interactive mode in which it will prompt the user to enter a command and wait for the user to respond. The commands that it will make available will allow the user to:

- Get the frequency distribution of the different atoms in the file,
- Get the frequency distribution of the different residues in the file,
- Get the maximum distance in angstroms between all pairs of atoms in a specific residue,
- Quit the application.

This assignment will give you more experience working with Perl hashes, processing text files, and writing subprograms. The details of the assignment are specified in the Detailed Requirements section below.

Background

A PDB file contains information obtained experimentally, usually by either X-ray crystallography, NMR spectroscopy, or cryo-electron microscopy, about a macromolecule. The PDB file completely characterizes the molecule, providing 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 or which nucleotides if it is a poly-nucleotide such as DNA or RNA, and much more. The information is not necessarily exact; associated with some of this information are confidence values or other measures that indicate the degree to which the researchers believe that the information is accurate.

Sometimes the researchers who created the file were not sure which of several measured positions of atoms to use, and rather than making a decision, they supplied several different models of the molecule’s structure. PDB files with multiple models are easy to spot because they have lines that begin with the word MODEL. Your program can be designed with the assumption that there is only a single model in the file; it does not have to be concerned with the possibility of there being multiple models in it. If a file has multiple models, there will be multiple lines that start with the word MODEL followed by a model number.

A protein can be made up of multiple chains. A chain is a linear sequence of amino acid residues. The same amino acid residue can occur multiple times within a single chain. Therefore in a PDB file, each residue in a chain is given a sequence number that specifies its position in the chain, starting with 1 as the first position.

Some PDB files also have multiple records that represent the same atom within a single model because the researchers who created the file were not sure which of a few measured positions of atoms to use, and instead of creating separate models, they put different choices of position for these atoms. Your program can be designed with the assumption that every atom has just a single record. When an atom has more than one position, it is identified by a specific character in the ATOM record in the file.

There are two kinds of atom records in a PDB file: ATOM records and HETATM records. ATOM records describe the atoms in the molecule itself. HETATM records are used to describe atoms that are not part of the biological polymer, such as those in the surrounding solvent or in attached molecules. In this project we ignore all atoms described by HETATM records.
An **ATOM** record contains several fields, specified by column numbers on the line. Your program will not need all of the information in the record; the information that your program does need is located on the line according to the following PDB file specification:

<table>
<thead>
<tr>
<th>COLUMNS</th>
<th>DATA TYPE</th>
<th>FIELD</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 6</td>
<td>Record name</td>
<td>&quot;ATOM &quot;</td>
<td>Atom serial number.</td>
</tr>
<tr>
<td>7 - 11</td>
<td>Integer</td>
<td>serial</td>
<td>Atom name</td>
</tr>
<tr>
<td>13 - 16</td>
<td>Atom</td>
<td>name</td>
<td>Atom name</td>
</tr>
<tr>
<td>17</td>
<td>Character</td>
<td>AltLoc</td>
<td>Alternate location indicator</td>
</tr>
<tr>
<td>18 - 20</td>
<td>Residue name</td>
<td>resName</td>
<td>Residue name</td>
</tr>
<tr>
<td>22</td>
<td>Character</td>
<td>chainID</td>
<td>Chain identifier</td>
</tr>
<tr>
<td>23 - 26</td>
<td>Integer</td>
<td>resSeq</td>
<td>Residue sequence number</td>
</tr>
<tr>
<td>31 - 38</td>
<td>Real(8.3)</td>
<td>x</td>
<td>Orthogonal coordinates for X in Angstroms</td>
</tr>
<tr>
<td>39 - 46</td>
<td>Real(8.3)</td>
<td>y</td>
<td>Orthogonal coordinates for Y in Angstroms</td>
</tr>
<tr>
<td>47 - 54</td>
<td>Real(8.3)</td>
<td>z</td>
<td>Orthogonal coordinates for Z in Angstroms</td>
</tr>
<tr>
<td>77 - 78</td>
<td>LString(2)</td>
<td>element</td>
<td>Element symbol, right-justified</td>
</tr>
</tbody>
</table>

The above table specifies which columns the data is in, what data type it is, and what it represents. For example, the serial number is up to five digits long and is in columns 7, 8, 9, 10, and 11, and the atomic coordinates are fixed decimal real numbers with 3 digits of decimal precision, with x, then y, then z in that order on the line. Atom names can be up to four characters long. For example, carbon atoms that are part of a ring are named CA, CB, CG, and so on, for C-alpha, C-beta, and C-gamma respectively.

**Detailed Requirements**

The program must use the file named on the command line as its input file. For example, if the program is named pdbtool, then if the user types

```
pdbtool 1A36.pdb
```

the program must read the data in the file named 1A36.pdb. If the file argument is missing, or if the file does not exist or cannot be opened for reading, the program should quit with an appropriate error message. The program should be robust enough that if the file is not a PDB file, it will not crash. This will require no extra effort if the program is designed to look only for the **ATOM** records in the file. The program can assume that all **ATOM** records are in the correct format; it does not have to check that the lines follow that format.

The program should read the data in the PDB file, and for each distinct atom in the file, store a record for that atom in an array of atom records. The program can assume that the file contains only a single model. If an **ATOM** record contains a non-blank character in column 17, it is an alternate location for an atom already read and should be skipped; the program will use only the first position found. For each atom, the program will need to store:

- The atom's serial number (cols 7 - 11)
- The three-letter name of the amino acid (called the residue) to which it belongs (e.g., ARG) (cols 18 - 20) in uppercase
- The one-letter chain to which this atom belongs (column 22), in uppercase
- The integer residue sequence number of the residue in which this atom is located (columns 23 - 26)
- The atom's three coordinates (x,y,z) (cols 31 - 54)
• The atom’s one- or two-letter element name (e.g. C, O, N, Na) (cols 77-78), first letter uppercase, second in lowercase if present.

After reading all of the data the program must display a message indicating how many atom records were created, such as

5285 atoms recorded.

The program will prompt the user to enter a command, with a short and simple prompt such as

command:

The user will be allowed to enter one of the following commands:

atomfreq
resfreq
reslength residue_name chain_id residue_seq_num
quit

If the user enters the command “atomfreq” the program displays, for each distinct atom that was stored, a line of the form

element:  n

where element is the element name and n is the number of atoms of that element. The lines should be sorted alphabetically by element name. Sample output could be

C: 3201
N: 918
O: 1101
P: 42
S: 23

If the user enters the command “resfreq” the program displays, for each distinct residue that was stored, a line of the form

residue:  n

where residue is the three-letter residue name and n is the number of occurrences of that residue in the file, sorted alphabetically. Sample output could be

ALA: 32
MET: 42
GLU: 91
...

If the user enters the command

reslength res_name chain_id res_seq_num
where res_name is a three-letter residue name in uppercase and chain_id is a one-letter chain id and 
res_seq_num is an integer that uniquely identifies the position of the residue named res_name in the given 
chain, the program should calculate the distance between every pair of atoms in this residue and display the 
maximum distance of these distances. If the command entered were

   reslength GLU A 3

the program should output the information in the following format:

   GLU with sequence number 3 in chain A has length 42.00 angstroms.

Finally, the quit command terminates the program with no resulting output.

Program Considerations

The program will need several different hashes. An atom record should be stored as a hash for example, 
and the array would be an array of references to these hashes. We have not yet learned about user-defined 
functions, but if we can create functions to simplify your main program. This is not a requirement.

Parsings the ATOM records can be easy or hard depending on how you do it. You cannot rely on counting 
how many words are on the line, which will vary. You must get the data from the exact set of columns 
specified above. Therefore, my suggestion is that your program read an entire line and convert it to an array 
of characters. Then it can use array subscripts (or perhaps the array slice operator) to extract the exact 
subarrays that it needs and convert them back to strings. (Remember that column numbers above start at 
1 but arrays start at 0.)

The program should use functions to complete its tasks. The main program should be very simple – it 
should prompt the user and get the user’s commands within a loop. It should write an error message if the 
user does not enter a valid command. When the user enters one of the valid commands, it should call a 
function to compute whatever needs to be computed. The functions should be passed whatever arguments 
they need in order to perform their tasks. I suggest that you use functions such as the following, but it is 
only a suggestion.

1. A function named atom_frequencies, which, given an array of atom records, returns a hash whose 
keys are distinct element names and whose values are the frequencies of these atoms in the array.

2. A similar function named residue_frequencies that returns a hash whose keys are residue names 
and whose values are residue counts.

3. A function named length, which, given an array of atom records, returns the maximum distance 
between all pairs of atoms in that array.

This program will not be so hard if you remember that one way to dereference a reference variable is by 
putting it in curly braces. For example, if $arrayref is a reference to an array,

   ${$arrayref}[0]

is the first element in the array that is referenced by $arrayref. If $hashref is a reference to a hash with a key KEY, then

   ${$hashref}{KEY}

is the value associated with the key KEY. So how do you get to a specific key in an array of hashes?

As a reminder, the program must have a proper prologue containing the program name, authorship, date of 
creation, usage information, a description of the input that it accepts, what it does when given that input - 
what the output is.
Testing Your Program

All programs must be thoroughly tested before they are released to users. Create some sample small PDB files by editing the ones in the pdb_files directory. Manually figure out what the outputs should be or use a spreadsheet to do this if you are good at spreadsheet calculations. Run your program and make sure that your output matches the one you manually computed. I will create a few small files for you as well.

Submitting the Solution

This assignment is due by the end of the day (i.e. 11:59PM, EST) on Thursday, December 8, 2016. You are to submit it by entering the following command on any of the cs1ab hosts:

```
/data/biocs/b/student.accounts/cs132/bin/submitPerlproject 5 myproject5
```

where myproject5 is the name of your Perl program file. The program will create a copy of your file in the project directory. If it does so successfully, you will see a confirmation message on the screen. You will not be able to read this file, nor will anyone else except for me. You can list the contents of the directory to see your file using the `ls -l` command on

```
/data/biocs/b/student.accounts/cs132/projects/project5
```

If you decide to make any changes and resubmit, just run the command again and it will replace the old file with the new one.