BCB 444 Fall 2015 Project 3

Using Perl Hash to Collect and Count Data

Due Sep. 23 at 1 p.m.

This project will lead you into using Perl hash gradually. We will take a slower but steady pace so it's easier to learn this important feature in Perl. Because Perl hash is a unique feature of the Perl programming language, even if you know some other programming languages, you may still find it difficult to understand Perl hash sometimes. If any of the instructions below does not work on your computer or if you need more helps understanding them, ask the TA during the lab session or post your questions to the Blackboard Learn -> Discussions -> Project Discussion area so we may address your questions.

This project will solve (or *recycle*) the same problem you have solved in Project 1 using a Perl program find_item.pl in combination with the Unix commands tr, sort and uniq. This time, we are going to do the same thing with just Perl coding. Actually, we can easily do more things with Perl as you will see in the following.

Job 1: Data collection and counting

We will begin by modifying the original find_item.pl program so it will use hash to collect data from the file lactose.txt. Both files have again been attached to this project description for your convenience. Download them and transfer the two files to your account (be it on your own machine or on lab iMacs) if you don't have them there already, and then start with job 1.

Copy the find_item.pl program to a new file project_3_job1.pl and modify the new file to make use of Perl hash. Declare a hash table, say, "my %table;" above the while loop and replace the print command with a data collection line "$table{$1}++". Here you are inserting each collected item stored in the $1 special variable as a *key* to your %table hash, and increase the *value* associated with that key by one. Since the same key may appear multiple times in the input, you are essentially counting how many times each item shows up in lactose.txt file.

Below the while loop (beyond the ending right brace '}' symbol), add another line to report how many items have been collected from the lactose.txt file as follows:

```
print STDERR scalar keys %table, " unique items found\n";
```

This line prints out the number of unique *keys* collected in %table, which is also the unique item count. Because this information is auxiliary or informative, it is customary to send this information to STDERR so it can be easily separated from

actual data output. Because some items appear multiple times in the lactose.txt file (you learned about that in Project 1), to report the total number of items, add the following lines to your code below the print command to add them up:

```
my $count=0;
for (values %table) {
  $count += $_;
}
print STDERR "$count total items seen.\n";
```

This total count should match the line count output when you run the original find_item.pl on lactose.txt and pipe its output to the word count command wc:

```
perl find_item.pl lactose.txt | wc
```

What this job1 program does is to tell you that there are certain number of items collected from the input and how many unique items are among them. Submit your finished project_3_job1.pl program as the result for this job.

Job 2: Case conversion

In Project 1, we already noticed that many items have the same name and only differ in their letter case. To more correctly summarize the unique items, we should also unify the letter cases. You can use either the **lc** or the **uc** function in Perl to turn strings into all **l**ower **c**ase or all **u**pper **c**ase. Let's just turn them all into lower case as we did in Project 1.

Copy project_3_job1.pl into project_3_job2.pl, and then modify the program by converting the letter case of each item into lower case *before* you register it into %table. You should only need to change one line of the code by adding three new characters (including the **lc** command) to make this happen. If you modify your code correctly, your output should match the line count value reported in your Project 1:

```
perl find_item.pl lactose.txt | tr '[:upper:]' '[:lower:]' | sort |
uniq | wc
```

Up to this point we have replicated most of the Project 1 jobs related to lactose.txt with just one Perl program. What you probably do not know is that this single Perl program solution is much faster than the *piped* multi-program solution we have learned in Project 1. Linux pipes allow you to connect multiple programs to accomplish a computation task quickly if the task will only be performed once, but it is preferable to write *one* program that does it all if the task will be repeated multiple times or when the data it processes gets big.

Submit the project_3_job2.pl file as the result for this job. Let's see what more we can do with Perl.

Job 3: Per unique item summary

In Job 1 above you summarized the unique items as well as the total item count. In Job 2 you basically reported the same things except now same name items were merged together disregard their letter case. What if you want to know how many times a particular item appears in the input? Do you need to anything more to collect and report information such as the following?

```
agglutinin isolectin 2 precursor    1 times
probable solute-binding lipoprotein.      1 times
galactose-binding lectin precursor  1 times
lactase-phlorizin hydrolase precursor     3 times
alpha-lactalbumin precursor   17 times
lectin precursor  1 times
aldose 1-epimerase precursor  1 times
galactose/lactose-binding lectin i precursor    1 times
alpha-lactalbumin precursor (version 1)   1 times
agglutinin isolectin 1 precursor    1 times
lactose-binding protein precursor   2 times
```

The fact is, your project_3_job2.pl program already collected such information from lactose.txt onto %table! All you have to do is to output that information from the hash table. Copy project_3_job2.pl to project_3_job3.pl, and then modify the previous code that adds the count of each entry together, making it output individual item name and its count value instead:

```
my $count=0;
for (values %table) {
  $count += $_;
}
print STDERR "$count total items seen.\n";
```

Let us repeat the following. The correct data has been collected into %table and you only need to modify JUST THIS PART of the code to make it **print** out the individual item counts. Submit the modified project_3_job3.pl as your result for this job.

More hints for Job3:

- To print both a key entry and its count value, you can use the following:

  ```
  print "$key\t$table{$key} times\n";
  ```

- To loop through all entries in a hash %table, you have several options. Both of the following work:

  ```
  for my $key (keys %table) { … do printing here for each $key … }
  while (my ($key, $value) = each %table) { … do printing here … }
  ```

  Note that the second looping method gives you both the $key and $value data so you don't have to access $table{$key} for value anymore.

- You can remove the $count variable declaration and its STDERR printout command because it's no longer needed.

Job 4: Alphabetical ordering

In the output produced by Job 3, the list is not given in any particular order. This is very typical of hash looping: you are only guaranteed that each entry in a hash table will be visited *once* but *not in any particular order*. However, more often it is beneficial to users who will read the data output to have the data sorted in a particular ordering scheme. In this job, you are going to output the same information generated in job 3 but sort them alphabetically. Of course, there is the Unix sort command that you have used in Project 1 that could also be used for this purpose, but we want to learn how to sort within Perl directly in this job.

Copy project_3_job3.pl to project_3_job4.pl and modify it so it will produce the following sorted output. Submit this as your result for this job.

```
agglutinin isolectin 1 precursor    1 times
agglutinin isolectin 2 precursor    1 times
aldose 1-epimerase precursor  1 times
alpha-lactalbumin precursor   17 times
alpha-lactalbumin precursor (version 1)   1 times
galactose-binding lectin precursor  1 times
galactose/lactose-binding lectin i precursor    1 times
lactase-phlorizin hydrolase precursor     3 times
lactose-binding protein precursor   2 times
lectin precursor  1 times
probable solute-binding lipoprotein.      1 times
```

Job 4 hints:

- If you look at the output above, you will notice it is the keys (i.e., item names) that are sorted. So how about you first obtain all keys from %table and sort them, before you output the sorted individual keys with their associated count values?
- Although we have at least two looping methods over %table, the "while (…each %table) { … }" version won't work for this job because it returns one key-value pair at a time in no particular order. To sort the keys in a hash you want all keys returned at once so you may sort them!
- You should only need to modify one line in your existing code and only adding 5 characters to that line (including the word sort).
- Alternatively, you can obtain an array of all keys from %table, sort the array, then loop through the array to output the keys in alphabetical ordering. You can retrieve the count value for each key with $table{$key}.

Job 5: Rank ordering

If you are conducting actual research, likely you don't care so much about alphabetical ordering of your enzymes. Instead, you want to know which enzyme shows up more frequently. In our example, `alpha-lactalbumin`

`precursor` shows up 17 times! It deserves to be on top of your output list like the following:

```
alpha-lactalbumin precursor   17
lactase-phlorizin hydrolase precursor     3
lactose-binding protein precursor   2
agglutinin isolectin 2 precursor    1
probable solute-binding lipoprotein.      1
galactose-binding lectin precursor  1
lectin precursor  1
aldose 1-epimerase precursor  1
galactose/lactose-binding lectin i precursor    1
alpha-lactalbumin precursor (version 1)   1
agglutinin isolectin 1 precursor    1
```

Copy project_3_job4.pl to project_3_job5.pl, modify the job5 code so it prints like the above, and submit it as your result for this job.

Job 4 hints:

- You still need to obtain all keys from %table and sort them somehow, but now you are not sorting their name directly but through their associated count value.
- Remember the associated count value for each $key is $table{$key}.
- If you can sort items in one particular way and you want to reverse the ordering, just exchange $a and $b in your sort criteria (condition).
- You may only need to modify just one line of your code to sort in this way, but the modification is a bit more complicated.
- Creating an inverted hash (also known as dual hash) reversing the count -> key mapping does NOT work in this case because as you can see there are many different keys all have the same count value; the inverted hash will not have the same number of entries.

This last job is a bit harder to understand (but not actually harder to implement), so pay attention during the lab session and/or the Discussion forum on Blackboard Learn to see more hints we may be given out later about this job.