cs3157 – Advanced Programming
Summer 2006, Lab #1, 15 points
May 25, 2006

A first lab in perl...

Follow these step-by-step instructions. This lab must be submitted electronically (see instructions on class webpage, homework page) by Wednesday, May 31th, 4 pm.

Please feel free to submit it as early as you can. Later submissions will overwrite earlier ones (in case you find errors before the deadline).

Please note that the point of the labs is to teach and reinforce specific programming concepts, cut time off your homework assignment while trying to maintain your interest. If you find them tedious or pointless, please don't hesitate to bring it to your instructor's attention (the same applies the opposite).

If you decide to be creative (by all means) please make sure to note it in your **README** file. At minimum the **README** file should specify the name of each included file and a brief description of what they do. In addition, if you have any issues, please note them in the README file.

You are expected to use the perl debugger in any environment you feel most comfortable.

Step 0)

If you are using your laptop, please make sure to install cygwin and perl, this will allow you to simulate the unix environment from your laptop, so you can work offline. Else you can log in to your cs account/cunix.

- Create a cs3157 directory for all your labs and homework's for this class.
- cd into that directory and create a lab1 subdirectory for this lab.

For example:

```
bash# cd
bash# mkdir cs3157
bash# cd cs3157
bash# mkdir lab1
bash# cd lab1
bash# pwd
bash# ls -la
```

(where bash# is the unix prompt)

Note that the instructions below give you specific names for the files that you create. Hint for doing labs: Read through an entire step before you try to do it.
Have fun!!!

Step 1. (3 point)

Goal: basic perl usage.

Create a basic perl script which prints out a greeting, and asks the user for their first and last name as one string. It should then be able to print the following for someone who enters "first last":

1) Hello **first last**
2) Your initials are **F. L.**
3) Your name backwards is **tsal tsrif**

Call this script **hello1.pl** to allow it execute you must use the chmod command. (you can check permissions using the 'ls –la' in the current directory as we did in class).

```
bash# chmod +x hello1.pl
bash# ./hello1.pl
```

Note use the 'which perl' command to know which path to include at the top of your perl script.

```
bash# which perl
```

and it will respond with a fully qualified path, such as: `/usr/local/bin/perl`

Hint: See the Scalar variables section of the perl tutorial for assistance. Note that here is where the use of double versus single quotes is important. When you use double quotes around a variable (e.g., "$a"), then the print statement will interpret or evaluate the variable and print out its value. If you use single quotes around a variable (e.g., '$a'), then the print statement will NOT interpret or evaluate the variable and will simply print $a. Try it both ways and see what happens, but make sure that you submit it the right way (printing the value)!

Hint2: perldoc –f reverse

Step 2 (3 points)

Goal: some perl control logic

Copy your hello1.pl script to a file called **game1.pl** and make it run-able:

```
bash# cp hello1.pl game1.pl
bash# chmod +x game1.pl
```

Modify it so that it plays a simple game. The program will ask you to guess how many fingers it is holding behind it's back (assume a little gnome is inside and they only have 10 fingers)….and once you enter your guess, if it is correct, congratulates you, else if gives you a hint and allows you to guess one more time.

One more thing…..you want to time how long it took the play the game….the way to do it, is to grab the localtime at the beginning and at the end to calculate it….just calculate minutes and hours. (i.e it took 0 hours and 10 minutes to guess wrong ☺).

Use the built-in function `localtime()` to get the current time. You can read about this function if you follow one of the links on the references page to perl built-in functions. Basically, the syntax for the function is this:

```
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst)=localtime();
```

Note that you call the function without an argument to get the current time. The function returns an array, so you can call it as above, or like this:

```
@current_time = localtime()
```

and then look up the individual array entries (e.g., `$current_time[0]`) if that makes more sense to you (the entries are stored in the order listed, i.e., seconds is item 0).

Step 3. (5 points)

Goal: file manipulation and advanced perl practice for the homework

Create a perl script named **step3.pl** which will be used to read a text file and process it with specific goals in mind.. Here is one way of how to run through and print out a file:

```
open( FH, $somefile ) || die "uh-oh problem opening $somefile $!";
```

```
while ( <FH> ) { print "$_" }
close( FH );
```
This is different than dumping the entire file into an array, as sometimes we don't have enough memory to do the operation.

Modify the above so that it reads a file passed in as a command line argument.

We have two goals in mind

1) We want to count the number of unique words in the file.
2) We want to see what are the top 5 occurring words in the file (this is hard…see hint at end)
3) We want to count the ***average*** length of words in the file.
   Hint: Perl has a built-in function called length().

For testing purposed please use the following test file which is linked on the homework page:
`callw10.txt`

Do NOT submit this test file please!
Your program should prints out the information in a friendly way — i.e., not just a number.
Hope you remember how to read command line arguments (else see book/class notes).

Hint: so how would we count the top occurring words in your program?? The sort routine can take a block of code to use for comparisons (perldoc –f sort)….so if you collect the counts of each word in a hash, you would be able to say:
```
sort { $hsh{$b} <=> $hsh{$a} }
```

Please email me if this hint doesn't make sense….

Step 4. (4 points)

Goal: working with files and md5sums, for the hw and next lab.

Create a perl script named **step4.pl** which will be used to help you create a log in system to authenticate users.

You will take 3 arguments into the program

1. username
2. password
3. Filename to store it in.

If the file exists you will be appending to it (no need to check for duplicates (next week lab). Each line of the file should be the username followed by a colon, followed by the md5sum of the password
Example:
```
shlomo:a96a13b145eb57bf8dd14f7e0ea1e5a4
professor:8ca00feae7a4c7ced9ef1ea4aaeb197e
```

One way to check if you are doing it correctly is for example to create a file test.txt with the password in it, and call md5sum (linux program) as follows:
```
md5sum –t test.txt
```

Hint: http://search.cpan.org/~gaas/Digest-MD5-2.36/MD5.pm

Please see webpage for electronic submission instructions