CSE/ISE 337 Assignment 2
Due date: Tuesday, Mar. 22, at 11:55pm

Important! Must read: (a) When writing programs, you **must** use the techniques that are described in the lecture notes and illustrated in the given samples. You may **not** use methods, modules, packages that were not covered in lectures. (b) You **must** first read the lecture slides "0-Course-Overview.pdf" available in Blackboard – Documents – Lecture Slides, especially those related to assignments on Slides 0-9 to 0-14, and follow them. (c) Start working on this assignment right away; you will **not** be able to finish it if you wait until the last day. (d) You **must** have "`use strict;`" and "`use warnings;`" at the start of your programs.

1. (5pts) **Perl documentation**

   On the allv Linux machines, write one command for each below respectively to find
   (a) the version of Perl that your are running. What version is it?
   (b) the documentation of all command switches of the Perl interpreter "perl"
   (c) the documentation of all Perl built-in functions
   (d) the description of the function **`map`** in the Perl manual
   (e) the description of the predefined variable **`$!`** in the Perl manual

2. (9pts) **List operations**

   (a) What does the following command do when issued at the OS command line?

   ```
   perl -le "print map{('a'..'z')[rand 26]} 1..8"
   ```

   Based on this command write another similar OS command that can be used to generate 8-character long passwords. In each password, the first two characters are randomly chosen from the five symbols **`? ! - ; #`** , the next four characters are randomly chosen from the alphabetic letters in both upper and lower cases, and the last two characters are randomly chosen from decimal digits. Note that `map` produces an array, not string, and to concatenate arrays, you can use the `(@array1,@array2)` syntax.

   (b) What does the following Perl statement in a Perl script do?

   ```
   print join " # ", grep{$_ % 3 == 0} split / /, "1 3 4 5 7 8 9";
   ```

   Modify this statement so that it can be used to parse a course number in string such as "**`ISE337`**", extract all numbers, and print them. E.g., the output for "**`ISE337`**" would be "**`337`**". You may hard code the input string in your Perl statement.

   (c) Read the documentation for `reduce` in List::Util module. Then write a Perl statement that prints the sum of all odd numbers and the halves of all even numbers within [1, 100] inclusive. The output should be 3775. To load and import the reduce function into your program, add **`use List::Util "reduce";`** to the start of your program. For a reference on Perl modules, see http://learnperl.scratchcomputing.com/tutorials/modules/

3. (10pts) **Flow control structures**

   Write a Perl program that generates simple arithmetic quiz questions. It then prompts user for an answer and prints "Excellent!" if the answer entered is correct. For each question, the first operand is an integer randomly chosen from 1 to 20 inclusively. The second operand is another integer randomly chosen from 1 to 30 inclusively. The operation is randomly chosen

from addition and subtraction. The number of questions is obtained from one command line argument. After the quiz, a summary line is printed. A sample quiz session is shown below.

a) Your program must use the **`do ... until`** loop at least once
b) Your program must use *modifiers* (see Slide 3a-35) at least once
c) Your program must use **`unless`** at least once
d) To get a random integer between, e.g., 1 and 100, you can use **`int(rand 100)+1`**

```
You have total 5 challenges. Get ready. Start!
19 + 19 = 29

15 - 4 = 11
Excellent!

3 - 16 = -13
Excellent!

19 + 24 = 44

9 + 14 = 23
Excellent!

Your score is 3 out of 5. That's 60% correct. Congrats.
```

4. (8pts) **Files and file tests**

Write a Perl program to inspect all files that are specified in an input file, and to show if each specified file exists (denoted by **`E`**, or **`-`** if otherwise), is a textfile (**`T`**), is readable (**`R`**), is writable (**`W`**), or is executable (**`X`**). The input file contains names of the files to be inspected, one file name per line. The name of the input file is read from the command line, e.g., "`perl q4.pl input`", where the input file is called "`input`".

The format of the program output is illustrated below. Each inspected file occupies one line, with the five attributes and the file name shown in that order, tab-separated.

```
E     T     R     W     -     ../Q6-Proc-notes/output
-     -     -     -     -     ./hello.html
E     T     R     W     -     ../Q6-Proc-notes/proc.pl
E     T     R     W     X     test.pl
E     -     R     W     -     IMG_7187.jpg
```

5. (10pts) **Subroutines**

Important: you **cannot** yet use regular expressions in either part (a) or (b) of this question. You should use the ord() and chr() functions as we did in Assignment 1.

(a) Write a Perl program to solve Question 2 in Assignment 1 using subroutines and the `map` function. Define a subroutine that decrypts a single character and then use `map` to apply the subroutine to the entire cipher-text to do the decryption.

(b) Write another subroutine that does encryption instead of encryption, and then use the subroutine to encrypt the decrypted plaintext from part (a). Your encrypted text should match the input of that question.

6. (10pts) **Files and regular expressions**

In Question 7 of Assignment 1, we processed some real Web content obtained at http://adm-kng.com/archive. As before, a sample HTML source file is provided to you in the assignment folder, called "adm-kng.com.archive.html". Write a Perl script to obtain the URLs of all .jpg images from the page source, and then find all those images whose URLs start with 3x.media or 4x.media, where x can be any single decimal digit. Your script then generates a HTML file as output that contains all such images. A sample excerpt of the output is given below.

```
<html>
<title>337 A2 HTML images</title>
<body>
<img src="http://40.media.tumblr.com/66d8f19fd25c96f20b309da07da63def/tumblr_o2tfpqj6PI1qb7lg5o1_250.jpg">
<img src="http://41.media.tumblr.com/77a14801ac72d9bf4a2fa1aa8cb9f599/tumblr_o2d3szpXMd1qb7lg5o1_250.jpg">
<img src="http://40.media.tumblr.com/63bbfd219187e0c18b12563b61410558/tumblr_o2egcw0lDA1qb7lg5o1_250.jpg">
</body>
</html>
```

You may use regular expressions in this question, provided that all operators, patterns, and solution methods are from our lecture notes on Regular Expressions. Name your output file "**`images.html`**"

7. (8pts) **Hashes**

In Perl, the built-in hash **`%ENV`** contains environment variables from your shell. For example, the `OSTYPE` environment variable stores the type of operating system you are running. The `PWD` variable stores the current working directory. (a) What does the `PATH` variable store? (b) Write a Perl program that does the following tasks in a row:

- prints the number of environment variables that are in **`%ENV`**
- prints the content of the `PATH` variable in **`%ENV`**
- prints the number of directories in the `PATH` variable
- sorts the directories in the `PATH` variable, and prints them one directory per line
- adds the current directory to the `PATH`, and runs a Perl script in the current directory from within your program using **`system("while.pl");`**

To do the last task, you need to write a small program called `while.pl` that prints `"hello "` five times on a line. Add the shebang line to its beginning and make it an executable by following Slides 3-10 and 3-11.

The current directory is denoted by a single dot. You should take a look at the format of the `PATH` variable content, and add the current directory to it using that format.

8. (6pts) **References**

Write a Perl program that goes through all lines and all words of an input file, counts the number of occurrences of each unique word, and builds a hash of words. The keys in the hash are all unique words in the input file. The value of each key is a string consisting of N equal signs, where N is the number of occurrences of the key word in input. The program then prints the content of the hash in a tab-separated key: value format. For example, if the content of an input file is:

```
a foo a bar we will he will o bar a
a bar o bar a ll we
well o foo
```

Then the output of the program should be like the following:

```
ll:     =
```

```
a:      =====
he:     =
we:     ==
well:   =
bar:    ====
will:   ==
foo:    ==
o:      ===
```

We assume that the name of the input file is passed as one command line argument, and on each line of the input file, all words are space separated. You **must** use references to build a nameless hash.

9. (Bonus 4pts) Continue from Question 8 above. Build a 2-level hash. For each unique word *w* in the input file, instead of the string consisting of equal signs described above, construct a hash instead. The hash contains all words following *w* and the number of their occurrences following *w*, regardless of whether the two words are on the same line or across two lines. As in Question 8, the number of occurrences of each following word is still represented using a string of equal signs. After processing the input and building the 2-level hash, your program prompts the user for a word and displays the content of its "following-word" hash. E.g., using the sample input file in Question 8, for word "a", we have:

```
Enter a word: a
foo     =
bar     ==
a       =
ll      =
```

Deliverables

Your assignment submission should include two files: (a) a printout of all programs that you write. Concatenate them into one **plaintext** file called "a2-printout.txt". Each program should be clearly labeled with its corresponding question/part numbers. (b) A **zip** file that includes all individual programs that you write. Name it "a2-source.zip". Be sure to name each program using its question and part number, e.g., "q2part_a", "q2part_b", and so on. You should include certain amount of program documentation, i.e., in-line comments, in your programs for important steps used. Do not repeat what the line of code says; rather write comments to help readers to understand your code.

Total: 66 points

Submission instructions

The handing-in will be through Blackboard Assignment. The submission instructions are at: http://it.stonybrook.edu/help/kb/creating-and-managing-assignments-in-blackboard.
You **must** read the submission instructions very carefully, and check to make sure your assignment has been submitted correctly **before** the deadline.

You can only submit once! However you can save your work by clicking "Save" as many times as you like. Only click "Submit" after you have checked and are certain that all requirements are followed.

Late submissions will not be accepted. The due date is **11:55pm on Tuesday, March 22.**