# CSE/ISE 337 Assignment 2 – Perl Programming (Spring 2019)

Due Date: Sunday, March 31th, 2019, at 11:55 PM

**Instructions:**

(a) When writing programs, you **must** use the techniques that are described in the lectures. You may **not** use methods, modules, packages that were not covered in this course.

(b) Unless explicitly specified to do so in a question, you may not use regular expressions to respond to questions in Assignment 2. Regular expressions will be the focus of Assignment 3.

(c) Start working on this assignment right away; you will find it very difficult to finish if you wait until the last day.

(d) Please put the pragma **`use strict;`** and **`use warnings;`** at the start of all your Perl programs.

(e) Your code **must** run on a Computer Science department computer (either allv Linux machines or Windows 10 lab machines). Running only on your own computer will cost you up to 15 points.

1. **List Operations** (9 pts total)

(a) Explain what the following command does when issued at a command line?

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

Now modify this command in order to produce a new command that does the following:

1. Generate a string 12 characters in length.
2. Each character in the string is randomly chosen from alphabetic letters, in both lower and upper cases, decimal digits, and the following special characters: **`? ! - ; : # $ %`**

(b) Explain what the following Perl statement in a Perl script does?

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

Now modify this statement so that it can be used to:

1. Parse a colon, ":", separated string, extract all odd numbers, and
2. Extract all the odd numbers from the separated string, and
3. Print these odd numbers, sorted from largest to smallest, separated from each other a space.

E.g., given the input `"1:3:4:5:71:8:9"`, the output would be `"71 9 5 3 1"`. **Hint:** read the Perl documentation on the function `sort` to understand how to sort a list numerically.

(c) Read the documentation for the `reduce` function in the `List::Util` module using:

```
perldoc List::Util
```

Then define a Perl subroutine that takes a list of integers as input and calculates the sum of all the odd numbers and the halves of the even numbers in the list. E.g., if the input is the list `[1, 2, 3, 4, 5, 6]`, then the output would be `1 + 3 + 5 + 2/2 + 4/2 + 6/2 = 1 + 3 + 5 + 1 + 2 + 3 = 15`. Your subroutine must use `reduce` at least once.

To load and import the `reduce` function into your program, add:

```
use List::Util "reduce";
```

to the start of your program. Write a few Perl statements to call your subroutine with several lists to verify the correctness of your subroutine.

2. **Exponentiation** (9 pts)

Exponentiation: `f(x, y) = x`$^y$

Do **not** use the `**` operator to answer this question.

(a) Write a non-recursive Perl subroutine, which takes two integers, *x* and *y*, greater than or equal to zero, and computes the result of raising *x* the power of *y*.

(b) Write a recursive Perl subroutine, which takes two integers, *x* and *y*, greater than or equal to zero, and computes the result of raising *x* the power of *y*.

(c) Write a tail-recursive Perl subroutine, which takes three arguments, two integers *x* and *y*, greater than or equal to zero, and an accumulator, *ax*, and computes the result of raising *x* the power of *y*.

For each part of the question, write Perl statements to test the correctness of your subroutine. Please include large inputs in your test cases, describe whether any of your subroutines run into trouble on large inputs, and explain why or why not.

3. **Flow Control Structures** (10 pts)

Write a Perl program to help students learn prime numbers. The program will do the following:

1. The program keeps asking the user to enter an integer.
2. For each number the user enters as input, the program checks whether the number is prime.

3a. If the number is prime, then the program displays this information for the user.
3b. If the number is not prime, then the program displays all of the divisors for the number.

4. The program should only check numbers greater than 1.
5. If the user enters -1, then the program:
    - 5a. prints the total number of unique numbers tested, and
    - 5b. prints the total number of unique prime numbers seen,
    - 5c. and then terminates.

Note: that Perl has the functions `next` and `last`, which behave like the Python statements `continue` and `break`, respectively. These two functions may be useful in your program (though you are not required to use them). Read the Perl documentation to get more information about these functions. Sample execution and output for the program is given below:

```
> perl q3.pl
Enter an integer: 3
                         prime
Enter an integer: 9
                         3
Enter an integer: 24
                         2 3 4 6 8 12
Enter an integer: 1
                         Input must be an integer greater than 1
Enter an integer: 3
                         prime
Enter an integer: -1
                         unique numbers: 3, unique primes: 1.
```

4. **Files and File Tests** (10 pts)

Write a Perl program according to the following specification:

1. Input is a file whose name is specified as a command line argument to your program.
   ```
   > perl q4.pl q4-input
   ```
2. The input file contains the names of files, one per line.
3. Your program will read the filenames from the input file and inspect each of those files.
4. Your program should create and write to five files:
   - 4a. `efiles.txt` : contains the names of files that exist, one per line
   - 4b. `rfiles.txt` : contains the names of the files that are readable, one per line
   - 4c. `wfiles.txt` : contains the names of the files that are writable, one per line
   - 4d. `xfiles.txt` : contains the names of the files that are executable, one per line
   - 4e. `tfiles.txt` : contains the names of the files that are text files, one per line
5. Your program should print a summary of the results to `STDOUT`.

E.g., if the input file, `q4-input`, contains the following list of files:

```
/etc/shells
/bin/pwd
/vmlinuz
/usr/bin/xxd
./q4-input
```

For this example, the summary output should look like this.

```
5 existing files:   /etc/shells /bin/pwd /vmlinuz /usr/bin/xxd ./q4-input
4 readable files:   /etc/shells /bin/pwd /usr/bin/xxd ./q4-input
1 writable files:   ./q4-input
2 executable files: /bin/pwd /usr/bin/xxd
2 plain text files: /etc/shells ./q4-input
```

5. **Subroutines** (10 pts)

You **may not** use regular expressions in your answer to either part of this question.

(a) In DNA strands, base bond to each other as base pairs in the following ways:

1. Adenine (A) bonds to Thymine (T)
2. Cytosine (C) bonds to Guanine (G)

So, (3) describes a possible sequence of two strands of DNA bound together in a double helix, while (4) does not:

```
3.      ATAACGCGGTTC
        TATTGCGCCAAG

4.      ATAACGCGGTTC
        GCGGTATAACCT
```

Write a Perl subroutine called dna_checker that takes two sequences of DNA bases, represented as strings composed of "A", "T", "C", "G", and then determines whether the two sequences can bind into a double helix. Your subroutine should return True if the sequences can bind together, and False otherwise.

Write a Perl program to test your subroutine for correctness. Run your subroutine on several test inputs.

(b) Suppose we want to rank sentences from best to worst in the least rational way possible. Write a subroutine that does the following:

1. Input is a list of strings.
2. For each string calculate its value in the following way:
    - 2a. Convert each character in the string into its integer ascii value using the ord() function.
    - 2b. Compute the average for the sentence by summing up the integer ascii values of the characters in the string and then dividing by the number of characters in the string. The string average is the value of the string.
3. Reorder the list of strings by their values.
4. Return the reordered list. You have now ranked all the strings in the input from best to worst.

Write a Perl program to demonstrate the correctness of your subroutine. Make sure to run your subroutine on several interesting test inputs.

6. **Hashes** (10 pts)

Write a program that processes a set of database records stored in a file. The input file will contain 1 record per line. Each record contains 7 tab-separated fields, in this order: First name, Last name, Street number, Street name, City name, State, and Zipcode. Here is an example input, stored in a file called `a2q6-input.txt` distributed with the assignment.

```
Jimmy     Cook      65   Main St.       Stony Brook        NY   11799
John      Smith     30   Townsend Rd.   Port Washington    NJ   12345
Alice     Wong      10   Highland Park  Moriston           MI   98765
Kathy     Lee       118  Robinson Lane  Watertown          NJ   13579
```

Use a hash to count the number of records per state. Print the number of states contained in the database. Then, print the content of the hash after the file fully processed, one key/value pair per line. Then print the total number of records processed from the input file. An execution of your program on the sample input described above should look like this.

```
> perl q6.pl a2q6-input.txt
There are 3 separate states in the database.
MI 1
NJ 2
NY 1
Processed 4 records successfully.
```

7. **References** (10 pts)

Build a list of three hashes.

-The first hash stores three first-year courses
-the second hash stores three second-year courses
-the third hash stores three third-year courses.

For each course, the key would be the course number, such as “110”, and the value would be its course title, for example “Introduction to Computer Science”. Choose any courses that you like. Write a subroutine that does the following:

1. It takes three arguments:
    1a. The first is a reference to the list described above
    1b. The second is a number indicating the year, 1 for the first-year, 2 for the second year, and so on.
    1c. The third is a course number, e.g., 110.
2. It then checks the list of hashes to determine if it contains the given course in the given year
3a. If so, it returns the course title
3b. If not, it returns an alert message “Unknown course!”.

Finally, call the subroutine a few times on a variety of interesting inputs to verify its correctness.

**Submission Format (2 pts for the right submission format):**
Submit a “`a2_LastName_FirstName.zip`” file that includes the following:

-- A `README.txt` or `README.pdf` file. It should contain your name, ID, and CS machine used (Linux allv machines or Windows 10 lab machines). It should also contain the answers to all questions that require a written response. Please label each of your answers clearly with the question number and part for which they are a response.

-- For all the programs you write, each program should be in a separate .pl file. Please name each program file using its question number and part. For example, the program for your non-recursive exponentiation subroutine should be written in a file named "`2a.pl`".

**Total Points: 70**

**Submission Instructions:**
Assignments will be turned in through Blackboard. Make sure that your submission is correctly formatted before you turn it in, and that submission occurs before the deadline.

**You can only submit twice.** Only click "Submit" when you are sure you have followed all the submission requirements.

Late submissions will not be accepted. The due date is **11:55 PM, Sunday, March 31.**