Assignment 4

Due 4:00 PM on Monday, October 25

For full marks, you must use the design recipe when writing functions from scratch. In particular, be sure to use the types and conventions noted in the lecture notes and in class when writing your contracts. Test data for all questions will always meet the stated assumptions for consumed values.

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources. Please read the course Web page for more information on assignment policies and how to organize and submit your work. Be sure to **download the interface file** from the course Web page and to follow all the instructions listed in the Style Guide (on the Web page), in particular about the naming of files. Remember to use the public testing system.

Coverage: Module 5 (lists of atomic values)
Language level: Beginning Student Scheme
Restrictions:

- You may not use the Scheme functions `reverse` or `member` in any of your solutions.
- Note that since you are using Beginning Student Scheme, you may **not** use list abbreviations on this assignment.

Questions:

1. Using structural recursion, write a Scheme function called `reciprocate` that consumes a list and produces a list of numbers and symbols by reciprocating each entry in the list (the reciprocal of x is $1/x$). If the input list entry is 0, then its reciprocal is the symbol `'Infinity`. Likewise, if the list entry is `'Infinity`, then its reciprocal is 0. Finally, if the list entry is not a number or the symbol `'Infinity`, then it should be skipped.

   For example,
   `(reciprocate (cons 2 (cons 0 (cons 'Infinity (cons "boo" empty)))))`
   produces `(cons 0.5 (cons 'Infinity (cons 0 empty)))`.

2. Using structural recursion, complete the Scheme function `increasing-order?` that consumes a list of numbers, and produces `true` if the elements in the list are in increasing order. Increasing order means that the first element in the list is strictly less than the second element in the list, and so on. An empty list and a list with one number are both considered to be in increasing order. For example,
   `(increasing-order? (cons -2.3 (cons 0 empty)))` produces `true`, while
   `(increasing-order? (cons 2 (cons 2 (cons 5 empty))))` produces `false`.

3. Complete the Scheme function `swap-last-first` that consumes a string of the form "LastName, FirstName" and produces a string of the form "FirstName LastName". You may assume that the first name and last name each have at least one letter, and that a single space follows the comma. You may also assume that the input string has at most one comma. If there is no comma, then your function should produce the string `"no comma"`. For example,
   `(swap-last-first "Orchard, Jeff")` produces `"Jeff Orchard"` and
   `(swap-last-first "Potter, H")` produces `"H Potter"`.

   You will need a helper function to find the location of the comma in the consumed string. When completing helper functions, be sure to use structural recursion on the list of characters corresponding to the consumed string (and not directly on the string).

   You may **not** use any string functions other than `string-append`, `substring`, `string->list`, `list->string`, and `string-length`.

4. Using structural recursion, complete the Scheme function `long-strings` that consumes a list of strings, and produces another list of strings by selecting only those that are at least as long as the average length. An empty list should produce an empty list. For this question, you might find the list-consuming function `length` and the string-consuming function `string-length` useful.

   For example,
   ```
   (long-strings (cons "blah" empty))
   ```
   produces
   ```
   (cons "blah" empty)
   ```
   and
   ```
   (long-strings (cons "short" (cons "longer" (cons "longest" empty))))
   ```
   produces
   ```
   (cons "longer" (cons "longest" empty))
   ```