Homework 2

This homework is due Wednesday, Feb. 7 (in class).

The Scheme code for the following problems, clearly documented, in one file, and in order of the problems below, should be mailed to `coleman` on Atanasoff. Your solutions will be graded on their correctness, style, and documentation. Please test your functions; you may wish to send us a summary of your tests (but not all of the test data etc.).

Please program in an applicative style, using lists (not vectors). That is, for this homework you may *not* use the imperative constructs of Scheme, essentially any function with an exclamation point (!) in its name (such as `set!`, `set-car!`, and `set-cdr!`). You may assume that argument lists are not circular.

1. (5 points) Write a function `average` that takes a list of numbers and returns their average. Assume that the input is not empty.

2. (5 points) Write a function `prime?` that tests whether a natural number is prime.

3. (5 points) Write a procedure `deep-reverse` that takes a list as an argument and returns as its value the list with all its elements reversed and with all sublists deep-reversed as well. For example,

   ```
   (deep-reverse '((a b) (c d)))
   ```

   returns `((D C) (B A))`.

4. (5 points) Write a procedure `fringe` that takes a list as an argument, and returns a list whose atoms are all the atoms appearing in the original list or any of its sublists, arranged in left-to-right order. For example,

   ```
   (fringe '((a b) (c d)))
   ```

   returns `(A B C D)`.

5. (15 points) A set can be described by a membership predicate that determines whether an element is a member of the set. Allowing the user to construct a set from a membership predicate gives one the power to construct sets that may "contain" an infinite number of elements (such as the set of all prime numbers). Implement the following operations

   - `(empty-set)`, returns the empty set.
   - `(such-that p)`, returns a set containing just those elements x for which `(p` x`)` returns `#T`.
   - `(singleton e)`, returns a set containing the value of `e`.
   - `(union s1 s2)`, returns a set containing all x such that x is either an element of `s1` or of `s2`.
   - `(intersect s1 s2)`, returns a set containing all x such that x is both an element of `s1` and of `s2`.
   - `(complement s)`, returns a set containing just those x such that x is not an element of `s`.
   - `(has s e)`, tests whether `e` is an element of `s`.