Homework 3: More Scheme

Please turn in your code as a single file with the name homework3.scm

For these problems, you must use pure functional scheme. That is:

- You may not use a functions like **set!** whose name ends with '!'.
- You may not use **do**. All repetition must be via explicit recursion or the recursion implicit in **map** or similar functions.
- You may, however, use **define** – in fact you must use it.
- You may define additional helper functions.

Be sure to test your definitions using some implementation of the "R5RS" scheme standard, preferably the one in Racket – see Resources > scheme-links.html for a discussion of how to get it. (It is what I have been using in lecture.) If you use the Racket implementation of scheme, be sure that the language is set to R5RS – the name "R5RS" should appear at the bottom left corner of the window, and **not** something like "advanced student" or "Determine language from source".

1. Define (i.e. write) the function (echo lst). This function doubles each top-level element of list lst. E.g., (echo '(a b c)) returns (a a b b c c). (echo '(a (b c))) returns (a a (b c) (b c))

2. Define the function (echo-lots lst n). (echo-lots '(a (b c)) 3) returns (a a a (b c) (b c) (b c)), that is, it is like echo but it repeats each element of lst n times.

3. Define the function (echo-all lst) which is a deep version of echo. (echo-all '(a (b c))) should return (a a (b b c c)(b b c c)).

4. Define the function nth. (nth i lst) returns the ith element of lst. E.g., (nth 0 '(a b c)) returns a, and (nth 1 '(a (b c) d) returns (b c). You may assume that 0 ≤ i < (length lst). You may **not** use the functions list-tail or list-ref in defining nth.

5. Define a scheme function (assoc-all keys a-list) where keys is a list of symbols and a-list is an assoc-list. (An assoc-list is a list ((<key1> <value1>)(<key2> <value2>) …) whose elements are two-element lists (<keyi> <valuei>), whose first element is a key and whose second element is the associated value.) assoc-all returns a list of the data associated with elements of keys by a-list. E.g. (assoc-all '(a d c d) '((a apple)(b boy)(c (cat cow))(d dog))) returns (apple dog (cat cow) dog). Use map. Note that you can't simply use assoc as one of the arguments to map; you need to use a lambda expression.

6. Define a scheme function filter which takes two arguments: a function fn and a list lst. Filter should return a list consisting of those elements of lst for which the fn returns true. E.g., the value of (filter even? '(3 4 6 7 8)) should be (4 6 8)) (The function even? is a built-in function in scheme which returns #t if its argument is even and #f if odd.)