CS 257: Non-Imperative Programming: Scheme!
Homework 2 (Spring '09)

1. Exercises 4.6, 4.8, 4.18, 4.19, 4.20 from Springer and Friedman.

2. Write a function, *atom-depth*, which takes a list, *ls*, as its argument and returns a list where every atom is replaced by its depth in the list. Here, depth is defined to be the number of left and right parentheses which enclose the atom. For example, *(atom-depth '(a b c))* returns *(1 1 1)*, *(atom-depth '(a (b) c))* returns *(1 (2) 1)* and *(atom-depth '(((foo))))* returns *(((3)))*. Hint: Use a helper function with an extra parameter.

3. Write a function, *cond->if*, which takes a *cond* expression, and transforms it into a set of nested *if* expressions. For example,

```
> (cond->if '(cond ((> x y) (- x y)) ((< x y) (- y x)) (else 0)))
(if (> x y) (- x y) (if (< x y) (- y x) 0))
>
```

4. Write a function, *equal?*, which has the same behavior as the Scheme function with that name.

5. Write a tail-recursive function, *even-iota*, which takes an integer *n* as its argument and returns an ordered list of the even integers (in ascending order) greater than or equal to zero and less than or equal to *n*.

6. Write a tail-recursive function, *sin*, which takes a number, *x*, as its argument and returns $\sin(x)$. Your function should approximate $\sin(x)$ by summing the first 100 terms of the following Taylor series:

$$\sin(x) = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots$$

   Any helper functions you need should be defined within the body of *sin* using *letrec*. Note: There is a good way and a bad way to do this. The good way avoids computing the factorial which appears in the denominator of each term in the series from scratch each time.

7. The function *list?* takes an s-expression, *s*, as an argument and returns *#t* if *s* is a list and *#f* otherwise. Give a definition of the function, *list-all?*, which takes an s-expression, *s*, as an argument and returns *#t* if *s* is a list and all pairs in *s* are also lists. Otherwise, *list-all?* returns *#f*. For example, *(list-all? '(1 ((2 3)) (4)))* should return *#t* and *(list-all? '(1 ((2 . 3)) (4)))* should return *#f*. You may use *list?* to define *list-all?*.