CSSE 304 Day 11 (and perhaps 12)

1. Question from the previous class: Other procedures that can be shorter if written using list-recur. Instructor's example.

2. Question from the previous class: Which of the 5 procedures that we wrote using list-recur is a bad idea to write that way? Why?

Recall: Grammar for lambda-calculus expresions:

<LcExpr> ::=	<identifier> |	**variable use**
	(lambda (<identifier>) <LcExpr>) |	**abstraction**
	(<LcExpr> <LcExpr>)	**application**

3. Use this grammar to derive `((lambda (x) (x y)) z)`

Variable *x* **occurs free** in the LcExp *e* iff
one of the following is true:

F1. *e* is a variable, and *e* is the same as *x*.

F2. *e* is an abstraction (λ (y) e'), where *y* is different from *x* and *x* occurs free in *e*'.

F3. *e* is an application (e$_1$ e$_2$), where x occurs free in e$_1$ or in e$_2$.

Variable *x* **occurs bound** in the LcExp *e* iff
one of the following is true:

B1. *e* is an abstraction (λ (y) e'), where *x* occurs bound in *e*', **or** *x* and *y* are the same variable and *x* occurs free in *e*'.

B2. *e* is an application (e$_1$ e$_2$) where x occurs bound in e$_1$ or in e$_2$.

4. Free and bound occurrences of variables. Formal definitions of *occurs free* and *occurs bound (in box above)*
5. **Examples:**In each of the following expressions, does *x* occur free and/or occur bound?

- x
- t
- (x t)
- (lambda (x) (x t))
- ((lambda (x) x) x)
- (lambda (x) (lambda (t) (t x)))

Follow the grammar. **occurs-free?** is defined in EoPL

```
(define (occurs-bound? sym exp)
```

6. The **lexical depth** of a bound occurrence of a variable is the number of levels of nested `lambda`s and `let`s between this occurrence and the variable's definition. In `(lambda (x) (lambda (y) (x y)))`**,** the occurrence of *y* has depth 0 and the occurrence of *x* has depth 1.

7. The **lexical address** of a bound occurrence of a variable is a pair (d p), where d is that occurrence's lexical depth, and p is the variable's position within its "declaration list". The lexical address of a free variable includes the variable's name and an indication that it is free.

8. Example:

In **(lambda (x z)**
(lambda (y)
((x y) z)))
The occurrence of x has depth 1 and position 0.
The occurrence of y has depth 0 and position 0.
The occurrence of z has depth 1 and position 1.

9. Example of output from the lexical-address procedure that you will write:

```
(lexical-address '(lambda (a b c)
                    (if (eq? b c)
                        ((lambda (c)
                           (cons a c))
                         a)
                        b)))
```

➔

```
(lambda (a b c)
  (if ((: free eq?) (: 0 1) (: 0 2))
      ((lambda (c)
        ((: free cons)  (: 1 0 (: 0 0)))
       (: 0 0))
      (: 0 1)))
```

10. **Exercise:** What is the output from:

Note: We are only looking at syntactic properties now.
Don't worry about whether this code has a useful meaning in Scheme.

```
(lexical-address
 '((lambda (x y)
     (((lambda (z)
         (lambda (w y)
           (+ x z w y)))
       (list w x y z))
      (+ x y z)))
    (y z)))
```