# CS 257: Non-Imperative Programming: Scheme!
## Homework 5 (Spring '05)

## Part I

1. Exercises 7.2, 7.3, 7.6, 7.7, 7.8

2. Consider the following definition of *unary-map*:

```
(define unary-map
  (lambda (proc ls)
    (if (null? ls)
        ()
        (cons (proc (car ls))
              (unary-map proc (cdr ls))))))
```

    Rewrite *unary-map* so that it is tail recursive. Hint: Look at the tail recursive definition of *append*.

3. Write a function, *disjunction2*, which takes two predicates as arguments and returns the predicate which returns #t if either predicate does not return #f. For example:

```
> ((disjunction2 symbol? procedure?) +)
#t
> ((disjunction2 symbol? procedure?) (quote +))
#t
> (filter (disjunction2 even? (lambda (x) (< x 4))) (iota 8))
(1 2 3 4 6 8)
>
```

4. Now write *disjunction*, which takes an arbitrary number ($> 0$) of predicates as arguments.

5. A matrix, $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, can be represented in Scheme as a list of lists: *((1 2) (3 4))*. Without using recursion, write a function, *matrix-map*, which takes a function, $f$, and a matrix, $A$, as arguments and returns the matrix, $B$, consisting of $f$ applied to the elements of $A$, *i.e.*, $B_{ij} = f(A_{ij})$.

```
> (matrix-map (lambda (x) (* x x)) '((1 2) (3 4)))
((1 4) (9 16))
```

6. Using the function, *iterate*, and without using recursion, give a definition for the function, *iota*.

# Part II

Using the functions, *apply*, *select*, *map*, *filter*, *outer-product* and *iota*, and without using recursion, give definitions for the following functions:

1. *length* - returns the length of a list.

2. *sum-of-squares* - returns the sum of the squares of its arguments.

3. *avg* - returns the average of its arguments.

4. *avg-odd* - returns the average of its odd arguments.

5. *min* - returns the smallest of its arguments.

6. *shortest* - returns the shortest of its list arguments.

7. *avg-fact* - returns the average of the factorials of its arguments.

8. *tally* - takes a predicate and a list and returns the number of list elements which satisfy the predicate.

9. *assoc* - takes a key and a list of pairs and returns the pair with *car* equal to the key.

10. *list-ref* - takes a list and an integer, *n*, and returns the *n*-th element of the list.