☺ CS 3 Final Review Spring 2005 ☺

1. What does each of the following evaluate to in Scheme? If there is an error, write ERROR. For each which results in an error, what small change could you make to its arguments such that it does something reasonable?

a. `(se '(hello) 'there '())`

b. `(list '(hello) 'there '())`

c. `(append '(hello) 'there '())`

d. `((lambda (x y z) (cons x (cons y z))) 'z '(3) '())`

e. `(member (quote (a)) (append (list '(a b c)) (quote (a))))`

f. `(apply map '(square (2 3 4)))`

g. `(cadar (map reverse '(((1 2 3) 4) (5 6 7) (8 (9 10)))))`

h. `(accumulate - (every square 1234))`

2. Write a procedure, `count-examples`, that inputs a sentence of words and returns the number of appearances of the phrase “for example” in the text.

```
(for example this sentence contains for example twice)
```

Which recursive pattern does this procedure follow?

3. Write a function, `list-average`, that uses higher-order functions to find the average of the numbers in a list that may contain anything.

```
(list-average '(1 2 6 7)) ➔ 4
(list-average '(1 2 abc (3 4) 6 7 #f)) ➔ 4
```

4. Describe what `mystery` does.

```
(define (mystery funct)
   (lambda (x) (if (funct x) 1 0)))
```

b. Use `mystery` along with higher-order functions to write a procedure, `count-numbers`, that counts the number of items in a list that are numbers.

5. Sorting is a big deal in Computer Science. Earlier in the semester, we worked with an *insertion sort* procedure. It essentially starts with an empty sentence, and places each number from the input into its proper place until the input is empty. For reference, here is insertion sort:

```
(define (insert n l)
  (cond
    ((null? l) (list n))
    ((< n (car l)) (cons n l))
    (else (cons (car l) (insert n (cdr l))))))

(define (sort l)
  (if (null? l)
    l
    (insert (car l) (sort (cdr l)))))
```

Now we're going to work with a similar algorithm, called *selection sort*. It works, in a sense, opposite of insertion sort. It finds the smallest number in the input, removes it, places it in the output (after any other entries that have already been output), and repeats until the input is empty.

Suppose you are given a procedure called `remove`, that takes in a number and either a list or a sentence, removes the first occurrence of that number from that list/sentence, and returns the result. Now write `sort` according to the method described. You may use either lists or sentences, and you may use helper procedures.

6. Write a procedure, `deep-min`, that returns the least number in a general list that might consist of other lists.

```
(deep-min '((10 2 3) 5 (2 4 (1 8) 9) 4)) ➔ 1
```

7. Describe, qualitatively, what changes would be necessary to convert the *Difference Between Dates* code found in your reader to operate on a different calendar system, say a lunar calendar.

8. In a lab earlier this semester, we wrote a function that computed the n^th Fibonacci number. The Fibonacci sequence starts with two 1's, and each successive number is the sum of the previous two numbers in the sequence (so, it's 1 1 2 3 5 8 …). Here's the code for the function:

```
(define (fib n)
  (if (< n 2)
    1
    (+ (fib (- n 1)) (fib (- n 2)))))
```

You may have noticed that it performed reasonably well for small inputs, but took a ridiculous amount of time (and slowed the lab machines to a halt) when you gave it a number larger than, say, 30.

a. Why does this happen?

b. We would like to improve on the performance of our program. Write a procedure that does this. You may use helpers. (Hint: compute the 10th Fibonacci number by hand. How did you do it differently from how the function listed above does it? Hint #2: Use an accumulating recursion)