TENTH QUIZ

You have 15 minutes from the start of class to complete this quiz. Read the questions with care; work with deliberate speed. Don't give us more than we ask for. The usual instructions apply. Good luck!

Problem 1 (10 points)

Each of the functions below computes the average price of a list of restaurant structures (defined as usual).

```
(define RList-average-price1
  (lambda (RL)
    (local ((define total-price1
              (lambda (L n)
                (cond
                  ((empty? L) n)
                  (else (total-price1
                         (rest L)
                         (+ n (rrant-price (first L)))))))))
      (/ (total-price1 RL 0) (length RL)))))

(define RList-average-price2
  (lambda (RL)
    (local ((define total-price2
              (lambda (L)
                (cond
                  ((empty? L) 0)
                  (else (+ (rrant-price (first L))
                           (total-price2 (rest L))))))))
      (/ (total-price2 RL) (length RL)))))

(define RList-average-price3
  (lambda (RL)
    (local ((define total-price3 (foldr + 0 (map rrant-price RL))))
      (/ total-price3 (length RL)))))
```

As you answer the following questions, consider each version of `RList-average-price` (including its local definitions). You may just answer 1, 2, and/or 3 for each question.

(a) (2 points) Which of the three versions use explicit recursion?

(b) (2 points) Which of the three versions use an accumulator?

(c) (2 points) Which of the three versions will require linear space on the call stack?

(d) (2 points) Which of the three versions are tail-recursive?

(e) (2 points) Which of the three versions use functions as arguments?

Problem 2 (10 points)

Suppose we run a department store with four departments (men's, women's, housewares, cosmetics), numbered 0 through 3. The store is open seven days a week (numbered 0 through 6, with Sunday as 0). We can store the sales figures for each department on each day of a week with a two-dimensional table (a vector of vectors with a row for each department and a column for each day—we'll call this a "sales table") by calling

```
(define sales-this-week (create-table 4 7))
```

using this definition:

```
;; create-table: number(departments) number(days) -> sales-table
;; Return a two-dimensional table with the specified number of rows and columns, all 0
(define create-table
  (lambda (num-departments num-days)
    (build-vector num-departments (lambda (n) (build-vector num-days (lambda (i) 0))))))
```

This creates a table full of zeroes. We can retrieve a given sales figure from the table (which will be more interesting once we insert some values—see below), in this case sales in the women's department (number 1) on Tuesday (day number 2) by calling

```
(sales-figure sales-this-week 1 2)
```

using this definition:

```
;; sales-figure: sales-table number(department) number(day) -> number
;; Return the value stored for the specified department and day
(define sales-figure
  (lambda (table department day)
    (vector-ref (vector-ref table department) day)))
```

We can record a sale of $137.50 in the housewares department (number 2) on Friday (day number 5) by calling

```
(add-sale sales-this-week 137.50 2 5)
```

using this definition:

```
;; add-sale: sales-table number(amount) number(department) number(day) -> sales-table
;; Return the table with the specified element increased by the amount
(define add-sale
  (lambda (table amount dept day)
    (vector-set!
      (vector-ref table dept) day (+ (sales-figure table dept day) amount))))
```

(a) (2 points) Write a Scheme expression to return the sales figure for the men's department on Saturday.

(b) (2 points) Write a Scheme expression to record (in the table `sales-this-week`) a sale of $295.00 in cosmetics on Thursday.

(c) (6 points) Complete the definition of the function below, which records returned merchandise. This is very similar to a definition above.

```
;; return-sale: sales-table number(amount) number(dept) number(day) -> sales-table
;; Return the table with the specified element decreased by the amount
(define return-sale
  (lambda (table amount dept day)
```