SEVENTH QUIZ

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

Brief reminders: `even?` returns true if its input is an even number; `positive?` returns true if its input is greater than zero; `add1` adds one to its argument; the function `member?` behaves as follows:

```
;; member?:  X  list-of-X  ->  boolean
;; Return true if the item occurs on the list
```

Problem 1 (5 points)

What is the value of each of the following expressions?

(a) `(map add1 (list 2 4 6 8 10 12))`

(b) `(map (lambda (s) (string-append s "!")) (list "Huey" "Dewey" "Louie"))`

(c) `(filter even? (list (* 5 3) (- 4 3) (* 5 2) (/ 350 10) (add1 1)))`

(d) `(filter (lambda (n) (and (positive? n) (even? n))) (list -4 -3 -2 -1 0 1 2 3 4))`

(e) `(foldr + 0 (list 40 50 60))`

(f) `(build-list 5 (lambda (i) i))`

(g) `(build-list 4 add1)`

Problem 2 (5 points)

Suppose we have a list called AL of animals (in a nature preserve) defined as follows:

```
(define-struct animal (name species age weight food))
```

where name and species are strings, age and weight are numbers (in years and kilograms, respectively), and food is a string representing the animal's favorite food.

For each of the following expressions, describe in one clear and precise English phrase what value it returns. Don't just say, "It does a foldr of plus and zero and ..."; give a description of what the expression *means*, something you could put in a software catalog so that a prospective buyer could find what he or she wanted. Use real-world terms, not program syntax terms: Say something like, "a list of the names of the gorillas who weigh over 100 kg," not "animals where species equals gorilla and whose animal-weight field is greater than 100."

(a) `(filter (lambda (A) (member? (animal-species A) (list "Lion" "Tiger" "Bear"))) AL)`

(b)
```
(local ((define L (map animal-age
                       (filter (lambda (A) (string=? (animal-food A) "oats"))) AL)))
  (/ (foldr + 0 L) (length L)))
```

Problem 3 (4 points)

Using `map`, `filter`, and/or `foldr` as necessary, define the following function without using explicit recursion. [Hint: You may use `local` to define intermediate results.]

```
;; shipping-weight-for-species:  list-of-animal  string  -> number
;; Return the total weight of all animals of the specified species.
(define shipping-weight-for-species
  (lambda (L s)
```

Problem 4 (6 points)

A binary search tree is either empty or a node defined as `(define-struct node (rootvalue leftsubtree rightsubtree))`.

(a) (2.5 points) Complete the following definition by filling in the blanks, one function, constant, or other identifier per blank:

```
;; BST-traverse-inorder:  BST  ->  list
;; Traverse BST "in order", collecting all items into a list
(define BST-traverse-inorder
  (lambda (T)
    (append
      (_____________________ (_____________________ T))

      (list (_____________________ T))

      (_____________________ (_____________________ T)))))
```

(b) (3.5 points) Complete the following definition by filling in the blanks, one function, constant, or other identifier per blank. To make this function work for any type of root value, we pass it a function that compares two items in the tree to see if the first is less than the second.

```
;; BST-member?:  item   BST-of-item  (item item -> boolean)  ->  boolean
;; Return true if item occurs in the tree
(check-expect (BST-member? 3 (make-node 2 empty (make-node 3 empty empty)) <) true)
(define BST-member?
  (lambda (i T item<?)
    (cond
      ((empty? T) false)
      ((equal? i (_____________________ T)) true)

      ((_____________________ i (_____________________ T))

        (_____________________ i (_____________________ T) item<?))

      (else (_____________________ i (_____________________ T) item<?)))))
```