COMP 210, Fall 2000
Second Exam
Due Friday, October 27, 2000 at 5:00pm
Hand the exam in at DH 2065

You will need to supply your own paper.

This exam is conducted under the Rice Honor Code. It is a closed-notes, closed-book exam. You have **two hours** to take the exam. You may take one break of up to fifteen minutes during the exam. The break does not count against your two hours. Of course, you are not allowed to discuss the exam with other people during the break, or to consult your notes, book, or DrScheme.

You may use a computer to type your answers to the various questions. You may not use DrScheme during the exam [or any other Scheme interpreter or compiler].

Please

1. Print your name legibly on the outside of the blue book. If you use more than one blue book, print your name on the outside of each blue book.
2. Write the pledge *correctly* and sign it. (No points this time.)
3. Record your starting and stopping times.

The exam has four questions. The relative weights (points) are marked on each question. If a question uses the results from an earlier question, you should assume that you have the perfect solution to that earlier question. For example, if question one has you create a program **foo** , you can assume, in question two that **foo** exists and that it works as specified, *even if your answer to question one is incomplete or incorrect.*

You may use the usual Scheme constructs and constants in your programs.

I have tried to give you enough information to answer each question on the exam. In the event that you need additional information, or a clarification, make a reasonable assumption and document it. (Write down, explicitly, any assumptions that you make.)

1. ***Programs that manipulate trees*** (25 points)

   In this problem, you may not use **filter**, **map**, **foldl**, or **foldr**.

   Consider the following definition for a parent-centric family tree. It is similar to the one that we used in lecture 13 and in the homework.

   ```
   ;; a parent is a structure
   ;;        (make-parent   name gender  children)
   ;; where name is a symbol, gender is a symbol (either ‘male or ‘female), and
   ;;         children is a list-of-children
   (define-struct parent (name gender children) )

   ;; a list-of-children is either
   ;;        — empty, or
   ;;        — (cons  f  r)
   ;;  where f is a parent and r is a list-of-children
   ;;  [We will use the built-in list constructor cons to implement this one]
   ```

   Develop the program **majority** that consumes a parent and returns a symbol. If the total number of males in the tree exceeds the total number of females, then **majority** should return the symbol **‘men**. If the total number of females exceeds the total number of males, then **majority** should return **‘women**. If the numbers are equal, then **majority** should return **’equal**.

   Show *__all__* *the steps in the development of your program.*

2. ***Programs that use two complex arguments*** (25 points)

   In this problem, you may not use **filter**, **map**, **foldl**, or **foldr**.

   Develop the program **unique**. It consumes two lists of numbers and produces a new list of numbers. The new list contains every number that occurs in either list, with no duplicates.

   Assume that the input lists are strictly ordered–-that is, the list elements are in ascending numerical order and there are no duplicates within an input list. (The same number may appear in both input lists, but no number appears twice in the same input list.)

   Show *__all__* *the steps in the development of your program*
   .

3. **Local** (25 points)

   a. (15 points) Given the following definitions

```
(define (fee x)
  (local [(define y 2)]
     (local [(define z 13)]
        (* x y z))))
```

   Show, by hand evaluation, what happens when you ask DrScheme to evaluate the expression

   (fee 3).

   Show all the steps in the rewriting and renaming process.

   b. (10 points) In class, we saw two versions of the program to compute the largest element in a list of numbers. They are:

```
;; max1 : nelon -> number
;; Purpose: returns the largest number in the input non-empty-list-of-numbers
(define (max1  a-nelon)
  (cond
     [(empty? (rest a-nelon))   (first a-nelon)]
     [(cons?  (rest a-nelon))
        (cond
           [(> (first a-nelon)  (max1(rest a-nelon)))
            (first a-nelon)]
           [else (max1 (rest a-nelon))]
        )]

   ))

;; max2 : nelon -> number
;; Purpose: returns the largest number in the input non-empty-list-of-numbers
(define (max2  a-nelon)
  (cond
     [(empty? (rest a-nelon))   (first a-nelon)]
     [(cons?  (rest a-nelon))
      (local  [(define maxrest   (max2 (rest a-nelon) ) ) ]
              (cond
                  [(> (first a-nelon)  maxrest) (first a-nelon)]
                  [else maxrest ] ))
        ]
    ))
```

   Explain how the behavior of these two programs differs. In particular, explain how many times each function is invoked in a typical call, like

   (max1 (list 1 2 3 4)) or (max2 (list 1 2 3 4))

   How does using local change the program's behavior?

4. **Abstract functions** (25 points)

   a. (15 points) Write the contract, purpose, and header for DrScheme's built-in abstract functions **filter**, **map**, **foldl**, and **foldr**

   b. (10 points) Following the contract that you wrote for it in part a of this question, implement the function **map**.

      Show the final code, including contract, purpose, and header.

Extra Credit (10 points)
Extend the program that you wrote in question 2 so that it correctly handles duplicates in the input lists. (Hint: use a helper function to keep your solution clean.)