COMP 210, Spring 2001, Homework 6
Due Wednesday, February 28, 2001 at the start of class

Before you start the homework, you should remind yourself of our General Advice, Advice on Homeworks, and Grading Guidelines. All are available from the class web site (http://www.owlnet.rice.edu/~comp210).

You do not need to write out data definitions for the various **`list-of`** data types. By this point in the class, we assume that you can do that part of the problem in your sleep. You must, however, write down the contract, purpose, and header for every function that your write (including those in **local**s), provide test data for each function that is not in a **local**, show your templates, and show your testing.

1. (3 pts) Evaluate (by hand) the following Scheme expressions. Show each step in the rewriting process.

   a. Given
   ```
   (define (fa x)
             (local [(define x 1)] x))
   ```
   Evaluate **`(fa 3)`**

   b. Given
   ```
   (define (fb x)
             (local [(define y 2)(define z 3)]  (* x y z )))
   ```
   Evaluate **`(fb 3)`**

   c. Given
   ```
   (define (fc x)
             (local [(define y 2) (define z 3)]
                 (local [(define y 4)] (* x y z ))))
   ```
   Evaluate **`(fc 3)`**

2. (3 pts) Develop a function **`sort`** that consumes a list of numbers and produces a list containing those numbers sorted into ascending order. Your sort function should use helper function, **`merge`** and **`split`** that inserts a number into a sorted list of numbers. Use **`local`** to make **`merge`** and **`split`** available only inside **`sort`.** The merge function was covered inLecture 12 on *Files, Directories, and Folders*. The template for the sort function can be found at

   `http://www.owlnet.rice.edu/~comp210/Assignments/split.scm`

   Turn in a complete solution including all of the steps in the design recipe except data definitions for **`list-of`** data types. Hint: you should initially write the program without using **`local`**, so that each function can be individually tested and debugged.

3. (4 pts.) Recall the definition of directories from Lecture 12. Write a function **`directory=?`** That takes two directories as arguments are determines if they are equal. To be equal, two directory trees must look exactly the same when they are drawn as trees. The only built-in equality operator in Scheme that you may use in this problem is **`symbol=?.`** You can check your answer by comparing the results produced by your function with the Scheme **`equal?`** function, which correctly determines the equality of **`directory`** objects (among other things).
4. Extra credit (4 pts.) Define a Scheme function **`perm`** that takes a list of symbols and returns a list of all the permutations of that list.