COMP 210, Fall 2000, Homework 6
Due Friday, October 20, 2000 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-* constructs. By this point in the class, we assume that you can do that part of the problem in your sleep. You must, however, writer 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.

.

a. (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 a helper function, *insert*, that inserts a number into a sorted list of numbers. Use *local* to make *insert* available only inside *sort*.

   Use the data definitions and templates for *list-of-number* to derive your solutions. (Write down the template; the data-definition for *list-of-number* is implicit.)

3. (4 pts) Develop a program *mergesort* as another way of sorting a list of numbers. Your program will use the function *merge* that we developed in Lecture 16 as a helper function.

   For this problem, it will help if you write out the data definition and the template for a list-of-list-of-numbers.

   Test each function independently, as you complete it.

   a. Develop a function that consumes a list of numbers and returns a list of one-element lists. That is, if it is given the input (list 5 4 3 2 1), it should produce the output

      (list (list 5) (list 4) (list 3) (list 2) (list 1))

   b. Develop a function that takes as input a *list-of-list-of-numbers* and repeatedly calls *merge* on successive pairs in the list. It should return a *list-of-list-of-numbers*. For example, invoking your function on the input

      (list (list 5) (list 4) (list 3) (list 2) (list 1))

      should produce the list

      (list (list 4 5) (list 2 3) (list 1)).

      Feeding that result back into your function would produce the list

      (list (list 2 3 4 5) (list 1)).

      Feeding this list back into your function would produce the list

      (list (list 1 2 3 4 5))

   c. Develop a function that calls the function from subpart (b) repeatedly, until the *list-of-lists-of-numbers* contains precisely one list.

   d. Finally, write *mergesort* using the functions from subparts (a), (b), and (c) as helper functions. *Mergesort* should consume a list of numbers and produce a list of numbers sorted into ascending order.