COMP 210, Spring 2000, Homework 5
Due Wednesday, March 15, 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. 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 each function, provide test data for each function, and show your testing (three to four tests per function, including trivial inputs).

1. (2 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. (2 pts) Write a function *member?* that takes a symbol *s* and a list of symbols *a-los* and returns *true* if *s* is in *a-los* and returns *false* otherwise. Note that *member?* should call itself recursively with two parameters.

   Now, rewrite *member?* to use a recursive helper function that takes only one parameter–-a list of symbols *l*. The parameter *s* to *member?* should not be passed explicitly to the recursive calls in the helper function. Use *local* in defining the helper function.

3. (2 pts) Write a function *sort* that consumes a list of numbers and returns the list of 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 the function *insert* only available locally. You should use the data definitions and templates to derive your solution. [Follow the templates!]

4. (4 pts) Develop a program *mergesort* as another way of sorting a list of numbers. Your program will reuse the *merge* program that we wrote in class on Monday. (See the notes for lecture 16, online.)

   Test each function individually, as you complete it.

   a) Write a function that takes as input a list of numbers and returns a list of one-element lists. That is, the input (list 5 4 3 2 1) should become

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

   b) Write a function that takes as input a list-of-list-of-numbers and repeatedly calls *merge* on successive pairs in the input list. It should return a list-of-list-of -numbers. For example, the input (list (list 5) (list 4) (list 3) (list 2) (list 1)) should produce the output (list (list 4 5) (list 2 3) (list 1)). Feeding the latter list into the function should produce (list (list 2 3 4 5) (list 1)), and using it again on this result should produce (list (list 1 2 3 4 5)).

   c) Write a function that calls the previous function repeatedly, until the list-of-lists contains no more than one item.

   d) Finally, write *mergesort* using these helper functions. It should take a list of numbers and return a sorted list of the same numbers.