CSE 3341 **Scheme Lab** **Due: Dec. 9, '22, 11:59 pm**

50 points

1. Do this lab using scheme48 either on your own machine or using an online interpreter. Do NOT use any other Scheme version. (If you use the CSE lab machine for doing this project, you may have to use the "subscribe" command to add scheme48 to your path.)

2. Do NOT use any Scheme primitives other than the ones we have discussed in class even if they are part of Scheme48. Specifically, do not use `lambda`, `let` and `if-then`.

3. **Note**: We haven't talked enough about *Scheme* for me to give you an interesting project in *Scheme*, so this is really a very simple (and, by itself, pointless) assignment. All it does is make sure that you get some practice in thinking functionally (which is a useful thing) and play with *Scheme* a little bit. In fact, even this lab may be a bit challenging, given the difference between programming in imperative languages such as Java or C++ and a functional language such as Scheme. Two key points to remember as you work on this lab are:
   i) Your functions (at least in design notation) should be fairly short;
   ii) Do not try to do more than one thing in one function.

4. If you want to get into *Scheme* in more depth, and it is worth getting into, the book *The structure and interpretation of computer programs* by Abelson and Sussman is highly recommended. It is available for free online.

Problem: The assignment is to write a function, `eliminateNsort`, which will receive two lists of integers as arguments. Suppose *L1, L2* are the two lists. The result returned by the function should be obtained from *L1, L2* as follows: start with *L1* and remove from it all those elements that are also in *L2*; then sort the remaining elements into non-decreasing order to get the result list.

The terminology in the last paragraph is incorrect. You do not actually want to *remove anything from L1*. You are doing this assignment *functionally* so values of variables are not supposed to change. (*Scheme* does have a `set!` operation which will let you change the values of variables; DO NOT use it; it is not meant for such problems. Also DO NOT use the *sort* function that is in the Scheme library.) Rather, the value returned by `eliminateNsort` will be a (new) list which will have all the elements of *L1*, except those that appear in *L2*, and, further, this list will be sorted.

Make sure that if some value *x* appears more than once in *L1*, then *x* appears the same number of times in the result returned by `eliminateNsort` unless *x* also appears in *L2* in which case it will not at all appear in the result.

Test your function with several combinations of *L1, L2* values (don't forget empty lists). You will find the "`,load`" command of the Scheme48 interpreter –I don't know if the online interpreters include "`,load`"– useful since it allows you to define a function in a file and then "load it in". You can get help on the load command (as well as some other scheme48-specific commands) by using the "`,help`" command of scheme48. Also useful is the "`,trace` command; it lets you 'trace' a function everytime it is called.

Approach: The trick to designing complex *Scheme* programs is of course to define a number of simpler functions and to do as little as possible in each. If you follow this rule, you will find this assignment reasonably straightforward to complete. If you don't follow the rule, you won't be able to finish the lab!

An even more important guideline: do not try to do two things in a function; if you do, you will likely get in trouble. So any time you find the need for computing something that is somewhat different from the things that the functions you already have (or the one you are currently defining) are capable of computing, introduce a *new* function rather than trying to add on that new functionality to the current function. This is a key difference between designing in a functional programming language vs. in an imperative language.

What to submit: You should submit a single file that contains the definitions of all your functions, including the main one `eliminateNsort`. The file itself should be called `myfns`. Do not use any other name for the file or for the main function. Other functions that you define may have whatever names you choose. `eliminateNsort` should be the last function defined in your file.

Use white space appropriately so your function definitions are easy to read. Include documentation in the *same* file (*not* a separate README file). Comment lines in *Scheme* programs start with a semi-colon.

Submission of the lab, as usual, will be on Carmen. And this detailed project statement will, again as usual, be on Piazza. If you have any questions or notice problems, please post on Piazza.