Homework 2: Procedures and Recursion

Due:

For problems that ask you to write procedures, hand in a printout of your code, and either a transcript showing how you tested each procedure with results, or an English argument that says why your code is correct or both. For other problems, you may handwrite a solution if you wish.

Transcripts should not be edited; it is better if you find problems with your code that you don't have time to fix than if we find them. English arguments should not be trivial (don't say "it looks right" or "I tested it and it works"), and it should not merely say what happens instead they should attempt to convince the reader that your program works correctly and can't be wrong on any input. As an example of a good solution, suppose we are to define the procedure `member?` that decides whether it's first argument is `equal?` to one of the top-level items in the list that is its second argument (see page 49). Here is a solution from page 50 of the text. The line starting with the semicolons is a comment that identifies what problem this is for your TA. Comments, anything following a "`;`" on a line, are ignored by Scheme itself.

```
;;; problem 2 (2.3.4 in the book)

(define member?
  (lambda (item ls)
    (cond
      ((null? ls) #f)
      (else (or (equal? (car ls) item)
                (member? item (cdr ls)))))))
```

The transcript for this problem might be as follows.

```
Chez Scheme Transcript [Tue Sep  1 21:52:10 1992]
> (load "member.ss")
> (member? 'x '())
#f
> (member? 'x '(x))
#t
> (member? 'x '(y z x q))
#t
> (member? 'x '(y z x))
#t
> (member? 'x '(a b c))
#f
> (member? 'x '((a) (b c) (x y z)))
#f
> (member? '(a) '((a) (b c) (x y z)))
#t
> (member? '(b c) '((a) (b c) (x y z)))
#t
> (member? 2 '(1 2 3))
#t
> (exit)
```

Note how this transcript tests different kinds of inputs: a symbol in the empty list, something at the beginning, middle and end of a list, and different kinds of data to be sure that the right equality test is used.

An English argument for the correctness of this procedure might go something like the following. As specified, `member?` uses `equal?` to test for elements being equal to the argument item. The argument `ls` must be a list, so it is either empty, in which case `item` is not in `ls` so the result is `#f`, or it is a list of the form `(cons` x l`)`. In the latter case, `item` is in `ls` if either `item` is equal to x = `(car ls)`, or `item` is in l = `(cdr ls)`. But we can assume (recursively, or as a mathematician would say, inductively) that `item` is in l = `(cdr ls)` if and only if `(member? item (cdr ls))` is true. It is well known in computing that English arguments like the above can be deceiving, and so it is best to test a little even if you are relying on these arguments for correctness. With practice it is possible to debug your programs by making such arguments and seeing where you feel uneasy about a step.

See the *Course Specification* handout for details of how we grade homeworks like this.

1. (2 points; basic) Do exercise 2.1.

2. (4 points; basic) Do exercise 2.3, both parts. You can handwrite this.

3. (4 points; basic) Do exercise 2.4.

4. (4 points; basic) Do exercise 2.5.

5. (8 points; basic) Do exercises 2.6 and 2.8. You can handwrite these.

6. (6 points) Do exercise 2.10.

7. (4 points) Do exercise 2.12. You can handwrite this.

8. (10 points; basic) Do exercise 2.13. Note that you are to define 3 procedures for this problem (see the end of the problem.

9. (12 points; basic) Do exercise 2.14.

10. (8 points; basic) Do exercise 2.15.

11. (8 points; basic) Do exercise 2.17.

12. (10 points) Do exercise 2.18.

13. (10 points; extra credit) Do exercise 2.19.

14. (10 points) Do exercise 2.20.

15. (15 points; extra credit) Categorize the kinds of functions that you have seen in this chapter. For each kind of function, describe the general form of the solution as an outline of a scheme function that would solve the problem. Compare your outlines with your solutions for the problems above and the programs in the book.

16. (15 points; extra credit) Categorize the kinds of mistakes that you have made in solving the problems in this chapter. For each kind of mistake, describe it and give an example along with a fix. Can you suggest ways to teach other students that would help avoid these mistakes?