Com S 342 Fall 2006 Name: _____

Principles of Programming Languages

Exam 1 on Language Design and Scheme Basics

This test has 8 questions and pages numbered 1 through 6.

Reminders

This test is closed book and notes.

If you need more space, use the back of a page. Note when you do that on the front.

This test is timed. We will not grade your test if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

For programs, indentation is important to us for "clarity" points; if your code is sloppy or hard to read, you will lose points. Correct syntax also matters. Check your code over for syntax errors. You will lose points if your code has syntax errors.

For Test Grading

Problem	Points	Score
1(a)	5	
1(b)	5	
1(c)	5	
2	5	
3(a)	5	
3(b)	5	
3(c)	5	
3(d)	5	
4	10	
5	10	
6	10	
7	15	
8	15	
total	100	

1. Consider the following features of the programming language Scheme.

   (1) Cond expressions, which evaluate one of several alternatives.
   (2) Lambda expressions, which have formal parameters and a body.
   (3) Numbers, with literals, and operations such as `+` and `-`.
   (4) Lists, which can hold any number of elements.
   (5) Symbols, which represent names and have a fast equality test.
   (6) The define special form, which can name values for later use.

   For each of the following, list the features above that best fit into the question's category. Don't list a feature in more than one category. If there are no such features, write "none".

   (a) (5 points) Which of the above are "means of computation"?

   (b) (5 points) Which are "means of combination"?

   (c) (5 points) Which of the above are "means of abstraction"?

2. (5 points) Why do Scheme programs look like lists? Briefly explain in terms of Scheme's design goals.

3. This is a problem about using procedures like `car`, `cdr`, `caar`, etc. Consider the following Scheme definition.

```
(define haiku
  '((wild sea)
    (stretching to Sado Isle)
    (the Milky Way)
    (from (The Narrow Road to Oku) (by Basho))))
```

For each of the following, your answer should depend on the value of `haiku`. Thus, for example, a quoted datum like `'Basho` is not correct. You may not use `list-ref` in your answers.

(a) (5 points) Write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the list `(wild sea)` from `haiku`.

(b) (5 points) Write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the symbol `sea` from `haiku`.

(c) (5 points) Write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the list `(Sado Isle)` from `haiku`.

(d) (5 points) Write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the symbol `Milky` from `haiku`.

4. (10 points) Draw a box-and-pointer diagram for the following list.

```
((wild sea) (the Milky Way))
```

5. (10 points) Using only parentheses, the procedure `cons`, quoted symbols (such as `'this`), and the empty list, `'()`, write a Scheme expression that produces the list.

```
((wild sea) (the Milky Way))
```

6. (10 points) Write a Scheme procedure, `salute`, with type

```
(-> ((list-of symbol) (list-of symbol)) (list-of (list-of symbol)))
```

which takes two lists, `professions` and `names`, each of which has exactly two symbols, and returns a list consisting of two sublists, the first sublist containing the first elements of `professions` and `names`, and the second containing second elements of these lists, in order. in that order. (Hint: this is *not* a problem about recursion, since you get to assume that there are exactly two elements in both arguments.) The following are examples.

```
(salute '(doctor professor) '(dobbs naumann))
   ==> ((doctor dobbs) (professor naumann))
(salute '(admiral private) '(hopper bailey))
   ==> ((admiral hopper) (private bailey))
(salute '(mayor senator) '(campbell harkin))
   ==> ((mayor campbell) (senator harkin))
(salute '(mother father) '(theresa time))
   ==> ((mother theresa) (father time))
```

7. (15 points) Write a recursive Scheme procedure `all-greater?`, of type

```
(-> (number (list-of number)) boolean)
```

that takes a list of numbers, `lon`, and a number `lower`, and returns `#t` just when each number in `lon` is strictly greater than `lower`. For example,

```
(all-greater? 3 '()) ==> #t
(all-greater? 5 '()) ==> #t
(all-greater? 5 '(4 5 6)) ==> #f
(all-greater? 5 '(5 6)) ==> #f
(all-greater? 3 '(6)) ==> #t
(all-greater? 8 '(27 9 6 15)) ==> #f
(all-greater? 9 '(-10 -12 -13 -500)) ==> #f
(all-greater? 10 '(11 11 11)) ==> #t
```

8. (15 points) Write a recursive Scheme procedure `symbol-or-los?`, with type

```
(-> (datum) boolean)
```

that takes a piece of Scheme data, `dat`, and returns `#t` just when dat is either a symbol or a list of symbols, and which returns `#f` otherwise. For example,

```
(symbol-or-los? 'yep) ==> #t
(symbol-or-los? '()) ==> #t
(symbol-or-los? '(a list of symbols)) ==> #t
(symbol-or-los? 342) ==> #f
(symbol-or-los? #t) ==> #f
(symbol-or-los? (lambda (x) x)) ==> #f
(symbol-or-los? '#(a vec)) ==> #f
(symbol-or-los? '((4 3))) ==> #f
(symbol-or-los? '(() 72 #f #(36))) ==> #f
(symbol-or-los? '((note that) () this (is not fine))) ==> #f
(symbol-or-los? '(most of these are symbols but not (this) one)) ==> #f
(symbol-or-los? '(all of these are symbols)) ==> #t
```

For this problem you are *not* allowed to use the `list-of` procedure. However, you can use Scheme's built-in `symbol?` and `list?` procedures to test if something is a symbol or a list.