Com S 342 Spring 2006 Name: _____

Principles of Programming Languages

Exam 1 on Language Design and Scheme Basics

This test has 7 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)	10	
3(d)	10	
4	10	
5	10	
6	15	
7	15	
total	100	

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

   (1) Named variables, which can hold values or references to objects.

   (2) Class definitions, which are named descriptions of objects and their methods.

   (3) Return expressions, which transfer control out of a method.

   (4) Blocks, which contain a sequence of statements and a final expression.

   (5) Objects, which can hold any number of fields and methods.

   (6) Character data, with literal expressions, and operations such as equality tests, comparisons, testing if lower case, etc.

   To answer the following, list all of the above feature categories of Scala that apply. If there are no examples above for the category of language features asked for in the question, 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) Scala supports XML literals, which make it easy to build XML documents in Scala programs. By contrast, Scheme's quotation mechanism makes it easy to build list structures and Scheme programs. What design goals of Scala and Scheme influenced this difference between the two languages? Briefly explain.

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

```
(define grook
  '(((problems) (worthy of attack))
    ((prove their worth) (by hitting back))))
```

For each of the following, your answer should depend on the value of `grook`. Thus, for example, a quoted datum like `'attack` 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 `(problems)` from `grook`.

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

(c) (10 points) Write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the symbol `attack` from `grook`.

(d) (10 points) Write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the list `(their worth)` from `grook`.

4. (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.

```
((problems) (worthy of attack))
```

5. (10 points) Write a Scheme procedure, `name-event`, with type

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

which takes a list `los` of exactly four symbols, and returns a list consisting of two sublists, the first sublist containing the first and second symbols of `los`, and the second containing the third and fourth symbols of `los` in that order, as shown in the following examples. (Hint: this is *not* a problem about recursion, since you get to assume that there are exactly four elements in `los`.)

```
(name-event '(lindsey kildow alpine skiiing))
   ==> ((lindsey kildow) (alpine skiiing))
(name-event '(carl swenson fifty-km cross-country))
   ==> ((carl swenson) (fifty-km cross-country))
(name-event '(irina slutskaya figure skating))
   ==> ((irina slutskaya) (figure skating))
(name-event '(apolo ohno short track))
   ==> ((apolo ohno) (short track))
```

6. (15 points) Write a recursive Scheme procedure `all-divisible-by?`, of type

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

that takes a number, `divisor`, and a list of numbers, `lon`, and returns `#t` just when each number in `lon` is divisible by `divisor`. You should assume that `divisor` is an integer that is strictly greater than 0 and that each number in `lon` is an integer. (Hint: you can use Scheme's `remainder` procedure to return the remainder of one number divided by another; for example `(remainder 17 2)` is 1.) For example,

```
(all-divisible-by? 3 '()) ==> #t
(all-divisible-by? 5 '()) ==> #t
(all-divisible-by? 5 '(4 50 55 -35 455125 55 90 10)) ==> #f
(all-divisible-by? 5 '(50 55 -35 455125 55 90 10)) ==> #t
(all-divisible-by? 3 '(3 27 9 6 15)) ==> #t
(all-divisible-by? 3 '(27 9 6 15)) ==> #t
(all-divisible-by? 3 '(3 27 1)) ==> #f
(all-divisible-by? 3 '(1)) ==> #f
(all-divisible-by? 7 '(5 4 2 1)) ==> #f
(all-divisible-by? 11 '(3 -27 0)) ==> #f
```

7. (15 points) Write a recursive Scheme procedure `list-of-lists?`, with type

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

that takes a piece of Scheme data, `dat`, and returns `#t` just when dat is a list in which each element is a list, and which returns `#f` otherwise. For example,

```
(list-of-lists? 342) ==> #f
(list-of-lists? 'nope) ==> #f
(list-of-lists? #t) ==> #f
(list-of-lists? (lambda (x) x)) ==> #f
(list-of-lists? '()) ==> #t
(list-of-lists? '((1 16) (4 3))) ==> #t
(list-of-lists? '((4 3))) ==> #t
(list-of-lists? '((72 98 36))) ==> #t
(list-of-lists? '(1 2)) ==> #f
(list-of-lists? '(())) ==> #t
(list-of-lists? '(() () () () (hmm) () () (ok) () ())) ==> #t
(list-of-lists? '((3 4) (5 4 3) (a b c d e f g) ())) ==> #t
(list-of-lists? '(no (3 4) (5 4 3) (a b c d e f g) ())) ==> #f
```

Hint: you can use Scheme's built-in `list?` procedure to test if something is a list.