Spring 2005
Com S 342

Name: _____
Section: _____

Principles of Programming Languages

Exam 1 on Language Design and Scheme Basics

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

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 Grading

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

1. Consider the following features of the string-processing language SNOBOL.

   (1) Numeric data, with operations such as addition, subtraction, etc.

   (2) Association tables, which can hold a mapping from one kind of data to another.

   (3) Patterns operators, such as alternation (p_1 `|` p_2), sequencing ($p_1 p_2$), that make new patterns from old.

   (4) Primitive patterns, such as `rem`, and `arb`, that are the basic building blocks of patterns.

   (5) Named variables, which can hold strings or patterns.

   (6) Named, programmer-defined functions.

   To answer the following, list all of the above feature categories of SNOBOL 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) In C and C++ arithmetic is defined in part by the implementation, so that, for example, an `int` might be either 16, 32, or 36 bits long. By contrast, Java specifies arithmetic in great detail; for example, in Java an `int` is a exactly 32 bits long and uses 2's complement notation. In Java, floating point arithmetic must follow the IEEE 754 standard in all implementations. What design goals of Java influenced this difference from C and C++? Briefly explain.

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

```
(define dictionary
  '((cool (that is good))
    (howdy (how are you doing))))
```

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

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

   (b) (10 points) Write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the list `(is good)` from `dictionary`.

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

```
(cool (that is good))
```

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.

```
(cool (that is good))
```

6. (10 points) Write a Scheme procedure, `delete-third`, with type

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

which takes a list of exactly four symbols, and returns a list consisting of the first, second, and fourth symbols in that order, as shown in the following examples. (Hint: this is *not* a problem about recursion.)

```
(delete-third '(one two three four))
   ==> (one two four)
(delete-third '(functional programming is great))
   ==> (functional programming great)
(delete-third '(this sentence has none))
   ==> (this sentence none)
(delete-third '(democratic freedom will flourish))
   ==> (democratic freedom flourish)
```

7. (10 points) Write a recursive Scheme procedure `symbols-first?`, of type

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

that takes a list of non-empty lists of scheme data, `ll`, and returns `#t` just when all the top-level elements of `ll` are lists that start with a symbol, and `#f` otherwise. You can assume that each inner list has at least one element. (Hint: you can use `symbol?` to test if a datum is a symbol.) For example,

```
(symbols-first? '()) ==> #t
(symbols-first? '((plus 3 4))) ==> #t
(symbols-first? '((minus 5 7) (plus 8 9) (3 plus 2) (times 3 4))) ==> #f
(symbols-first? '((plus 8 9) (3 plus 2) (times 3 4))) ==> #f
(symbols-first? '((3 plus 2) (times 3 4))) ==> #f
(symbols-first? '((times 3 4))) ==> #t
(symbols-first? '((hycos 27) (hsin 2) (cos 3) (sin 4) (tan 2))) ==> #t
(symbols-first? '((plus))) ==> #t
(symbols-first? '((plus) (times))) ==> #t
(symbols-first? '((72))) ==> #f
(symbols-first? '((+ is a symbol) (yes a plus-sign is a symbol))) ==> #t
(symbols-first? '((yes a plus-sign is a symbol))) ==> #t
(symbols-first? '((plus 3 4 2 5 4 1) (times 0 -2 5))) ==> #t
```

8. (15 points) Write a recursive Scheme procedure `all-length-n?`, with type

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

that takes a non-negative integer, `n`, and a list of lists of numbers, `llon`, and returns `#t` just when each sublist of `llon` has a length that is exactly equal to `n`, and `#f` otherwise. For example,

```
(all-length-n? 2 '()) ==> #t
(all-length-n? 2 '((1 16) (4 3))) ==> #t
(all-length-n? 2 '((4 3))) ==> #t
(all-length-n? 3 '((72 98 36))) ==> #t
(all-length-n? 3 '((72 98 36) (15 27))) ==> #f
(all-length-n? 0 '(())) ==> #t
(all-length-n? 15 '((1 2 3 4 5 6 7 8 9 10 11 12 13 14 15))) ==> #t
(all-length-n? 2 '((7) (0 0) (1 1) (3 4) (4 3) (3 4) (0 0))) ==> #f
(all-length-n? 2 '((0 0) (1 1) (3 4) (4 3) (3 4) (0 0))) ==> #t
(all-length-n? 2 '((1 1) (3 4) (4 3) (3 4) (0 0))) ==> #t
(all-length-n? 2 '((3 4) (4 3) (3 4) (0 0))) ==> #t
```

Hint: you can use Scheme's built-in `length` procedure in your solution.