Fall 2001 Name: _____
Com S 342 TA (or Section): _____

Principles of Programming Languages

Exam 1 on Language Design and Scheme Basics

This test has 11 questions and pages numbered 1 through 5.

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.

1. (5 points) What item in the following list is the best example of a "means of computation" in Scheme.

   (a) definitions, such as `(define colors '(red green blue yellow))`

   (b) applications, i.e., procedure calls, such as `(map car my-list)`

   (c) numbers, such as `3`, and operations on numbers, such as `+`

   (d) all of the above

   (e) none of the above

2. (5 points) What item in the following list is the best example of a "means of combination" in Scheme.

   (a) definitions, such as `(define colors '(red green blue yellow))`

   (b) applications, i.e., procedure calls, such as `(map car my-list)`

   (c) numbers, such as `3`, and operations on numbers, such as `+`

   (d) all of the above

   (e) none of the above

3. (5 points) What item in the following list is the best example of a "means of abstraction" in Scheme.

   (a) definitions, such as `(define colors '(red green blue yellow))`

   (b) applications, i.e., procedure calls, such as `(map car my-list)`

   (c) numbers, such as `3`, and operations on numbers, such as `+`

   (d) all of the above

   (e) none of the above

4. (10 points) In our class discussion, what design goals of Simple influenced the decision to have a syntax where expressions are written using a syntax like `(+ 3 (* 4 2))`? Briefly explain.

5. (5 points) As in our class discussion, how does dynamic type checking matches the design goals of Simple better than static type checking? Explain briefly.

6. (10 points) Given the following Scheme definition,

```
(define the-kingdoms
  '(animals (plants flowers) (bacteria germs) viruses archaea))
```

write a Scheme expression using procedures like `car`, `cdr`, `caar`, etc., that extracts the symbol `bacteria` from `the-kingdoms`. (Note: the expression you write should depend on the value of `the-kingdoms`, so `'bacteria` is not correct.)

7. (10 points) Draw a box-and-pointer diagram for the following list.

```
((oysters) (half shell))
```

8. (10 points) Using only parentheses, the procedure `cons`, quoted symbols (such as `'this`), and the empty list, `'()`, write a Scheme expression that produces the following list.

```
((oysters) (half shell))
```

9. (10 points) Write a Scheme procedure, `twist`, with type

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

which takes a list of exactly three symbols, and twists it one symbol to the right as shown in the following examples.

```
(twist '(a b c)) ==> (b c a)
(twist '(a good day)) ==> (good day a)
(twist '(fine trees grow)) ==> (trees grow fine)
```

10. (10 points) Write a Scheme procedure `all-upper-case?`, of type

```
(deftype all-upper-case? (-> ((list-of char)) boolean))
```

that takes a list of characters, `loc`, and returns `#t` just when all the elements of `loc` are upper-case characters, and `#f` otherwise. For example,

```
(all-upper-case? '()) ==> #t
(all-upper-case? '(#\J #\M #\L)) ==> #t
(all-upper-case? '(#\M #\L)) ==> #t
(all-upper-case? '(#\a #\b #\c)) ==> #f
(all-upper-case? '(#\J #\M #\L #\a #\s #\p #\e #\c)) ==> #f
(all-upper-case? '(#\J #\M #\L #\a #\s #\p #\e #\c)) ==> #f
(all-upper-case? '(#\space #\W #\A #\R #\P #\?)) ==> #f
(all-upper-case? '(#\W #\A #\R #\P)) ==> #t
```

Hint: use Scheme's `char-upper-case?` procedure, which returns `#t` just when its argument is a upper-case character.

11. (20 points) Write a Scheme procedure `equal-all?`, with type

```
(deftype equal-all? (forall (T) (-> ((list-of T)) boolean)))
```

that takes a list and returns `#t` just when all the elements of the list are `equal?` to each other, and `#f` otherwise. For example,

```
(equal-all? '()) ==> #t
(equal-all? '(a a a b)) ==> #f
(equal-all? '(a a a)) ==> #t
(equal-all? '(b a a a)) ==> #f
(equal-all? '(a a a)) ==> #t
(equal-all? '(a)) ==> #t
(equal-all? '(1 1 1 1 1 1 1 999 1 1 1)) ==> #f
(equal-all? '(1 2 3 4 4 3 2 1)) ==> #f
(equal-all? '(#(1) #(1))) ==> #t
(equal-all? (list (cons 1 '()) (cons 1 '()))) ==> #t
```

Hint: you may want to use a helping procedure.