CS212 Spring 2000
Final Exam
May 17, 2000

Name__________________________________________

Instructions

There are nine (9) numbered problems on this exam. Please check now that you have a complete exam with no missing pages. Write your name above. Be sure to try all of the problems, as some are more difficult than others. Don’t waste a lot of time on a problem that is giving you a hard time—move on to another problem and then return to it later. Read the questions very carefully; some of them are trick questions.

This exam is closed book. No papers, books or notes may be used, except for the Scheme/Swindle quickrefs, which will be provided. Please put your answers in the boxes provided. Extra paper is available if you need it. Proper indentation of your code is strongly encouraged.

Good luck!

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/8</td>
<td>/4</td>
<td>/12</td>
<td>/9</td>
<td>/6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>Σ</td>
</tr>
<tr>
<td></td>
<td>/10</td>
<td>/14</td>
<td>/11</td>
<td>/9</td>
<td>/83</td>
</tr>
</tbody>
</table>
The Adventures of Alice in Schemeland

Chapter 1: <pair>ing down the Rabbit-Hole (8 points)

Alice was beginning to get very tired of sitting by the computer in Upson hall, and of having nothing to do: once or twice she did a few problem sets for this CS212 course that she was taking, and what is the use of a CS course, thought Alice, without exams and finals?

So she was considering in her own mind (as well as she could, for the previous final exams made her feel very sleepy and stupid), whether the pleasure of making a make-chain! would be worth the trouble of getting up and writing the code, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be late!" but when the Rabbit actually took a Palm out of its backpack and used it, then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a backpack or a Palm organizer to put it in, and burning with curiosity of all things geek, she ran across the lab after it, and fortunately was just in time to see it pop down a large rabbit hole on the other side of the Consulting Office.

In another moment Alice was to go after it, but the door was locked. Instead she saw old cryptic runes, the likes of which looked vaguely familiar from the shapes and forms of her dreams while sleeping soundly in CS212...

HELP ALICE ENTER THE RABBIT-HOLE BY MATCHING EACH INPUT EXPRESSION ON THE LEFT WITH THE CORRESPONDING OUTPUT EXPRESSION ON THE RIGHT.

<table>
<thead>
<tr>
<th>(list 1 (cons 2 3))</th>
<th>(1 (2 . 3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(cons 1 (list 2 3))</td>
<td>(1 2 3)</td>
</tr>
<tr>
<td>(list 1 (list 2 3))</td>
<td>(1 (2 3))</td>
</tr>
<tr>
<td>(cons 1 (cons 2 3))</td>
<td>(1 2 . 3)</td>
</tr>
<tr>
<td>(cons (list 1 2) 3)</td>
<td>(((1 2) . 3)</td>
</tr>
<tr>
<td>(list (cons 1 2) 3)</td>
<td>(((1 . 2) 3)</td>
</tr>
<tr>
<td>(cons (cons 1 2) 3)</td>
<td>(((1 . 2) . 3)</td>
</tr>
<tr>
<td>(list (list 1 2) 3)</td>
<td>(((1 2) 3)</td>
</tr>
</tbody>
</table>
Chapter 2: Evaluate Me (4 points)

Down, down, down. Would the evaluation of \((\lambda x \ (x \ x)) \ (\lambda x \ (x \ x))\) ever come to an end? She was ready to terminate execution, when all of a sudden, it ended and thump! thump! thump! down she came upon a heap of Scheme manuals and intricately carved things created by madmen decades ago, and the fall was over.

Having landed in a huge bundle of s-expressions, Alice found herself covered and buried. “This is getting curioser and curioser!” cried Alice (she was much surprised). She came upon several s-expressions in little course packets labeled “Evaluate Me.” It was all very well to say “Evaluate Me,” but the wise little Alice was not going to do that in a hurry! There might be expressions with non-terminating conditions, or run-time errors. “No, I’ll think first,” she said, “and reason through what they should evaluate to.”

HELP ALICE EVALUATE THE PILE OF S-EXPRESSIONS BY SAYING WHAT THE RESULT OF EVALUATING EACH OF THE FOLLOWING IS:

(a) \((\text{head} \ (\text{tail} \ (\text{head} \ (\text{tail} \ (\text{head} \ (\text{tail} \ \text{head})))))))\)

\text{head}

(b) \(((\lambda x \ (+) \ 3) \ 4 \ 5)\)

9

(c) \(((\lambda \ (\text{car} \ \text{cdr}) \ (\lambda x \ (\text{if} \ x \ \text{car} \ \text{cdr}))) \ 1 \ 2) \ #t)\)

1

(d) \(((\lambda x \ (x \ x)) \ (\lambda x \ (x \ x)))\)

It doesn’t halt.
Chapter 3: The Mad #t Party (12 points)

Alice came upon a table set out under a binary tree in front of the lab, and Eli and Brandon were having tea at it: a Nadine was sitting between them.

The table was a large one, but all three were all crowded together at one corner of it: “No room! No room!” they cried out when they saw Alice coming. “There’s plenty of room!” said Alice indignantly, “you just haven’t garbage-collected memory,” and she sat down in a large fluffy futon at one end of the table.

“Have some wine,” the Eli said with an encouraging Israeli accent. Alice looked all round the table, but there was nothing on it but #t. “I don’t see any wine,” she remarked.

“There isn’t any,” said the Eli.

“Come, we shall have some fun now!” thought Alice. “I’m glad they’ve begun asking riddles—I believe I can guess that.”

“Do you mean that you think you can find a definition that will evaluate to 42?” said the Brandon.

“Exactly so,” said Alice.

“Then you should show us that you know what you mean,” the Eli said.

HELP ALICE DEMONSTRATE HER KNOWLEDGE OF SCHEME TO THE ELI AND BRANDON BY GIVING A DEFINITION OF ZARDOZ THAT WOULD CAUSE THE GIVEN EXPRESSION TO EVALUATE TO 42.
Give a definition of `zardo2` that would cause the given expression to evaluate to 42. If impossible, say so and explain. Example:

```
(define zardo2 '(42 43))
=> (head zardo2)
42
```

(a) 
```
(define zardo2 (lambda () (lambda (x) 42)))
=> (let ((zardo2 (zardo2)))
    (zardo2 zardo2))
42
```

(b) 
```
(define zardo2 4)
=> (let* ((zardo2 (+ zardo2 (- zardo2 1)))
    (zardo2 (* zardo2 (- zardo2 1))))
    zardo2)
42
```

(c) 
```
(define zardo2 (lambda (x) 42))
=> ((compose zardo2 zardo2) zardo2)
42
```

(d) 
```
(define zardo2 (list 42))
=> (begin
    (set! (tail zardo2) zardo2)
    (cadddr zardo2))
42
(Note: (cadddr ...) = (car (cdr (cdr (cdr ...)))))
```

(e) 
```
(defmethod (zardo2 (zardo2 <top>)) 42)
=> (defmethod (zardo2 (zardo2 <symbol>)))
  (+ (call-next-method) zardo2)
  (zardo2 2)
42
```

(f) 
```
(defmethod (zardo2 (zardo2 <top>)) 40)
=> (defmethod (zardo2 (zardo2 <number>)))
  (+ (call-next-method) zardo2)
  (zardo2 2)
42
```
Chapter 4: Off with their frames! (9 points)

At that moment, Nadine, who had been looking out across the table, called out “The Queen! The Queen!” and all three instantly threw themselves flat upon their faces. There was a sound of many footsteps, and Alice looked round, eager to see the Queen.

First came the global environment, and all the primitive types; next were the closures and local frames, and then the pointers and arrows.

When the procession came opposite to Alice, they all stopped and looked at her, and the queen said severely, “Who is this? What’s your name, child?”

“My name is Alice, so please your Majesty,” said Alice very politely, but she added to herself, “Why, they’re only a set of environments and closures, after all, I needn’t be afraid of them!”

“And what is this?” said the Queen, pointing to the three course TA’s at the table.

“What do you know?” said Alice, surprised at her own courage. “These are CS212 course consultants.”

The Queen turned crimson with fury, and, after glaring at her for a moment, screamed “Off with their frames! Off—”

Alice, who has been thinking this whole time (as she usually does), was curious as to where the Queen and her collection of frames has come from…

HELP ALICE FIND OUT WHERE THE QUEEN CAME FROM BY GIVING DEFINITIONS OF F AND G THAT RESULT IN THE FOLLOWING ENVIRONMENT.
(a) (7 points) Give definitions of \( f \) and \( g \) that would result in the following environment.

\[
\begin{array}{|c|c|}
\hline
f: & \text{params: (y)} \\
    & \text{body: (+ x y)} \\
    & \text{env: } \quad \bullet \quad \\
\hline
\end{array}
\qquad
\begin{array}{|c|c|}
\hline
g: & \text{params: (x)} \\
    & \text{body: (* x y)} \\
    & \text{env: } \quad \bullet \quad \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
x: & 11 \\
\hline
y: & 12 \\
\hline
\end{array}
\qquad
\begin{array}{|c|c|}
\hline
x: & 99 \\
    & y: 100 \\
\hline
\end{array}
\]

\[
\begin{array}{l}
(\text{define } f \\
    (let* ((x 11) (y 12)) \\
            (lambda (y) (+ x y))))
\end{array}
\]

\[
\begin{array}{l}
(\text{define } g \\
    (let ((x 99) (y 100)) \\
            (lambda (x) (* x y))))
\end{array}
\]

(b) (2 points) What would be the value of \((f \ (g \ 2))\) evaluated in the global environment after making these definitions? \((f \ (g \ 2)) \Rightarrow 211\)
Chapter 5: Who deadlocked the tarts? (6 points)

Having been brought on the good side of the Queen, Alice was taken back to the royal palace, where she was to witness a trial.

“Herald, read the accusation!” said the king.

On this the White Rabbit blew three blasts on the trumped, then uncurried the function, and read as follows:

The Queen of Threads chopped off some heads
And gave them to Brandon Bray
And if you have read about Scheme threads
You’d do well on the final today

“Call in the first witness,” said the King, and the White Rabbit blew three blasts on the trumpet, and called out, “(head (make-list *witnesses*))”

The first witness was the Vinocur. He came in with an RFC in one hand and a Palm III in the other. “So... I beg your pardon, your Majesty,” he began, “for bringing these in...”

“You ought to have checked news before you left,” said the King. “Who did it?”

The Vinocur looked at the Eli and the Brandon, who had followed him in. “Professor Kozen, in the Graduate Lounge, with the Knife, I think it was,” he said.

“But, eh, somebody killed his thread,” said the Eli, who had been watching all this time.

“But if all the threads were killing each other, then who stole the tarts?” said the Queen.

HELP ALICE SOLVE THE MYSTERY BY CONSIDERING THE FOLLOWING CONCURRENT PROGRAM AND COMMENTING ON MUTUAL EXCLUSION AND DEADLOCK.
Consider the following concurrent program:

\[
\text{(define critical-section)}
\]
\[
\hspace{1em} \text{(let ((mutex (make-semaphore 1)))}
\]
\[
\hspace{2em} \text{(lambda (name)}
\]
\[
\hspace{3em} \text{(semaphore-wait mutex)}
\]
\[
\hspace{3em} \text{(echo name "entering critical section")}
\]
\[
\hspace{3em} \text{(sleep (random 3))}
\]
\[
\hspace{3em} \text{(echo name "leaving critical section")}
\]
\[
\hspace{3em} \text{(semaphore-post mutex)}))\]
\[
\text{(define (compete name iterations))}
\]
\[
\hspace{1em} \text{(cond ((zero? iterations)}
\]
\[
\hspace{2em} \text{(kill-thread (current-thread))}}
\]
\[
\hspace{2em} \text{(else}}
\]
\[
\hspace{3em} \text{(critical-section name)}
\]
\[
\hspace{3em} \text{(sleep (random 3))}
\]
\[
\hspace{3em} \text{(compete name (- iterations 1)))))}\]
\[
\text{(let ((Bob (thread (thunk (compete "Bob" 5))))}
\]
\[
\hspace{1em} \text{(Carol (thread (thunk (compete "Carol" 5))))}
\]
\[
\hspace{1em} \text{(Ted (thread (thunk (compete "Ted" 5))))}
\]
\[
\hspace{1em} \text{(Alice (thread (thunk (compete "Alice" 5)))))}
\]
\[
\hspace{1em} \text{(sleep (random 10))}
\]
\[
\hspace{1em} \text{(kill-thread Ted))}\]

(a) (3 points) Is mutual exclusion achieved? Explain.

Yes, there is a single semaphore that is created when \text{critical-section} is defined, and all threads wait on this semaphore before entering the critical section, so at most one thread can be in the critical section at a time.

(b) (3 points) Is deadlock possible? Explain.

Yes, the main thread could kill Ted while Ted was in the critical section, and the semaphore would never get posted.
Chapter 6: Lexawocky (10 points)

T’was schemish, and the mutex loads
Did deadlock and compete in the lab
And flimsy was the pseudocode
And the problem sets were bad.

Beware the lexawock, my son!
The strings that null and proper prefix
We string-ref from character 0, not one
And compute strings in order lex

He took his Swindle sword in hand:
Longtime the functional code he sought
So consulted he at Upson 3
And came from lecture in thought...
Lexicographic order (or dictionary order) is an order relation on strings. The usual definition of lexicographic order is:

- If one string is a proper prefix of the other, then the shorter is lexicographically less than the longer. ("Proper prefix of" = "strict prefix of" = "a prefix of, but not equal to").
- If neither string is a prefix of the other, find the leftmost character position where they differ. The string with the smaller character code in that position is lexicographically less than the other.

An alternative inductive definition is:

- The null string is not lexicographically less than itself.
- The null string is lexicographically less than any nonnull string.
- If the first characters of $x$ and $y$ are $a$ and $b$, respectively, and the character code of $a$ is less than the character code of $b$, then $x$ is lexicographically less than $y$.
- If both $x$ and $y$ begin with the same character, and if $x'$ and $y'$ are the strings obtained by removing the first character of $x$ and $y$, respectively, then $x$ is lexicographically less than $y$ if and only if $x'$ is lexicographically less than $y'$.

Write a program to compute lexicographic order on strings. Use

(char<? a b) compare the character codes of a and b
(as <list> x) convert a string x to a list of characters

{(define (string-lex<? (x <string>) (y <string>))
  (letrec ((sl-aux
            (lambda ((a <list>) (b <list>))
              (cond
                ((null? b) #f)
                ((null? a) #t)
                ((char<? (head b) (head a)) #f)
                ((char<? (head a) (head b)) #t)
                (else (sl-aux (tail a) (tail b)))))))
             (sl-aux (as <list> x) (as <list> y))))}
Chapter 7: Through the Looking-Glass and into the stream (14 points)

After chasing the Lexawocky, Alice found herself in the middle of a large forest of trees, with a clearing in the middle.

“Oh my,” said Alice. “Where do I go from here?”

She stretched herself up on tiptoe, and peeped over the edge, and her eyes immediately met those of a large caterpillar, that was sitting on the top with its arms folded, taking not the smallest notice of her or anything else.

Over the top of the caterpillar, Alice saw a looking-glass. Filled with the curiosity that CS212 has taught her, she peered through the glass.

Alice looked into the looking-glass and saw a stream of natural numbers. “Such fresh and crisp natural numbers they must be,” she thought to herself. She tried to enter the stream but could not find a way in. Every time she tried to touch the stream, it would back away from her.

“Perhaps I need to find the complement of the stream,” thought Alice. Since the looking-glass was inverting the image, she might be trying to add herself to an infinite set for which she does not satisfy the predicate…

HELP ALICE GET PAST THE STREAM BY WRITING A FUNCTION THAT TAKES A STREAM S AND PRODUCES ITS COMPLEMENT.
Suppose \( s \) is a stream of natural numbers such that

- the numbers in \( s \) occur in strictly increasing order;
- there are infinitely many numbers that do not occur in \( s \).

Write a program that takes such a stream \( s \) and produces its complement; that is, the stream of all numbers that do not occur in \( s \). You may use the following without definition:

- \( \text{stream-head } s \) get the first element of a stream
- \( \text{stream-tail } s \) get the remainder of the stream
- \( \text{cons-stream } x s \) make a new stream with head \( x \) and tail \( s \)

```
(define (stream-complement (s <stream>))
  (letrec ((sc-aux
    (lambda (n s)
      (if (= n (head s))
        (sc-aux (+ n 1) (tail s))
        (cons-stream n (sc-aux (+ n 1) s))))))
    (sc-aux 0 s)))
```

```
(define multiples-of-3
  (filter (lambda (x) (zero? (remainder x 3))) naturals))
```

```
(print-stream (stream-complement multiples-of-3))
=> (1 2 4 5 7 8 10 11 13 14 16 17 19 20 22 23 25 26 28 29 ...)
```

```
(print-stream (stream-complement primes))
=> (0 1 4 6 8 9 10 12 14 15 16 18 20 21 22 24 25 26 27 28 ...)
```

(Note: The functions \( \text{filter, print-stream} \), etc. for streams were defined in lecture, as were the class \( \text{stream} \) and the streams \( \text{primes, naturals} \), etc. \( \text{filter} \) for streams works much the same way as it does for lists. The complexity of your solution will be considered in the grading. We also discourage the use of side effects.)
Chapter 8: Selectable, delectable cats (11 points)

Alice crossed the stream and trod quietly into the woods on the other side. She came to a split in the road, where the road diverged off into many possible paths. How could she decide which way to go? She was just saying to herself, “if one only knew how to solve the Halting Problem—” when she was a little startled by seeing the Cheshire Cat sitting on a bough of a ternary tree a few nodes off.

The Cat only grinned when it saw Alice. It looked well-formed, she thought: still it had very long asymptotic runtimes and a great many helper functions, so she felt that it ought to be treated with care.

“Cheshire Puss,” she began, rather timidly, as she did not at all know whether it would like the name: however, it only grinned a little wider.

“Come, it’s pleased so far,” thought Alice, and she went on. “Would you tell me, please, how to select which way I ought to go from here?”

“That depends a good deal on which list element you want to select,” said the Cat.

“I don’t want unwanted side-effects—” said Alice.

“Then you don’t want to evaluate every element,” said the Cat.

“—so long as I evaluate one s-expression,” Alice added as an explanation.

“Oh, you sure can do that,” said the Cat, “if only you had another special form…”

HELP ALICE CREATE A NEW SPECIAL FORM “SELECT”

In this problem you are to add a new special form to the tiny-scheme interpreter from PS6. You may find the following definitions useful (some definitions have been simplified from the problem set):

```
(define (ts:error (msg <string>))
  (echo "error:" msg)
  (*ts:continue-continuation*))

;;;;; ts:eval

(define (ts:eval (exp <top>) (env <env>)))

;;;; default: return the expression as the value
```
(defmethod (ts:eval (exp <top>)) (env <env>)) exp)

;;; evaluate a symbol by lookup
(defmethod (ts:eval (var <symbol>)) (env <env>))
  (let ((bv (env-lookup-binding env var)))
    (if bv
      (binding-value bv)
      (unbound-var env var))))

;;; evaluate a combination (e1 e2 e3 ... en)
(defmethod (ts:eval (combination <pair>)) (env <env>))
  (ts:eval-combination (head combination) (tail combination) env))

;;; ts:eval-combination

;;; evaluate a combination (operator e1 e2 e3 ... en)
;;; sub-exps is the list of expressions (e1 e2 e3 ... en)
(defgeneric (ts:eval-combination operator sub-exps (env <env>)))

;;; default: evaluate the operator and arguments, then
;;; call ts:apply to apply operator value to argument values
(defmethod (ts:eval-combination operator sub-exps (env <env>))
  (ts:apply (ts:eval operator env)
    (map (lambda (e) (ts:eval e env)) sub-exps)))

;;; special form (lambda (x1 ... xn) e)
;;; create a closure
(defmethod (ts:eval-combination (operator = 'lambda) sub-exps
  (env <env>))
  (make <closure>
    :closure-args (first sub-exps) ; (x1 ... xn)
    :closure-body (second sub-exps) ; e
    :closure-env env))

;;; special form (if e1 e2 e3)
(defmethod (ts:eval-combination (operator = 'if) sub-exps
  (env <env>))
  (if (ts:eval (first sub-exps) env)
    (ts:eval (second sub-exps) env)
    (ts:eval (third sub-exps) env)))
The new special form is called `select`. The first argument evaluates to a nonnegative integer and the remaining arguments are expressions. `select` evaluates the first argument, and then selects the corresponding expression (counting from 0). The selected expression is evaluated and the value is returned. None of the other expressions are evaluated.

```lisp
==> (select 0 56)
56
==> (select 2 (/ 1 0) (+ 2 2) (+ 3 3))
6
==> (select (+ 1 1) 45 67 12 65)
12
==> (select 56 12 13 14)
error: selector value is out of range
==> (select -1 67 78 99)
error: selector value is out of range
==> (select "hello" "there")
error: first argument must evaluate to an <integer>
==> (define (foo n) (select n (/ 1 0) 2 (/ 3 0)))
==> (foo 1)
2
```

Write the code that must be added to the interpreter to handle `select`.

```lisp
(defun ts:compile-combination (operator = 'select) exps)
  (when (< (length exps) 2)
    (ts:compiler-error (cons 'select exps) "Too few arguments"))
  (let ((n (ts:compile (head exps))))
    (if (integer? n)
      (if (and (>= n 0) (< n (length (tail exps))))
        (ts:compile (list-ref (tail exps) n))
        (ts:compiler-error (cons 'select exps)
                         "Selector out of range"))
      (cons 'select (cons n (tail exps))))))

(defun ts:eval-combination (operator = 'select) exps (env <env>))
  (let ((n (ts:eval (head exps) env)))
    (if (integer? n)
      (if (and (>= n 0) (< n (length (tail exps))))
        (ts:eval (ts:compile (list-ref (tail exps) n)) env)
        (ts:eval-env (cons 'select exps) "Selector out of range")
                    (ts:eval-env (cons 'select exps) "Selector not an integer"))))
```

16
Chapter 9: Tweedledee, Tweedledum, and Tweedledoo (9 points)

Alice came across the three of them, and Alice knew in a moment which was which for two out of the three, because one of them had a “DUM” embroidered on his collar, and another had “DEE.” I suppose they’ve each got “TWEEDLE” round at the back of the collar, she said to herself.

“If you think we’re wax-works,” he said, “you ought to pay, you know. Wax-works weren’t made to be looked at for nothing. Nohow!” “I’m sure I’m very sorry,” was all Alice could say; for the words of the old song kept ringing through her head like the ticking of a clock, and she could hardly help saying them out loud: —

Tweedledum and Tweedledee
Inherit from each other
For Tweedledum and Tweedledee
Had Tweedledoo, another.

So if you know what to define
And functions to defmethod
To get the output that you will find
When using call-next-method

“I know what you’re thinking about,” said Tweedledum: “How can you get our behavior?”

HELP ALICE SORT OUT TWEEDLEDEE, TWEEDLEDUM, AND TWEEDLEDOO BY DEFINING THEIR CLASSES TO GET THE FOLLOWING BEHAVIOR.
(a) (7 points) Define the classes <tweedledee>, <tweedledum>, and <tweedledoo> so as to get the following behavior.

```lisp
(defclass <tweedledee> ()
  (defmethod (frumious (bandersnatch <top>))
    (echo "brillig"))
(defclass <tweedledum> ()
  (defmethod (frumious (bandersnatch <tweedledee>))
    (echo "slithy")
    (call-next-method))
(defclass <tweedledoo> (<tweedledee> <tweedledum>))
(defgeneric (frumious bandersnatch))
(defmethod (frumious (bandersnatch <top>))
  (echo "brillig"))
(defmethod (frumious (bandersnatch <tweedledee>))
  (echo "slithy")
  (call-next-method))
(defmethod (frumious (bandersnatch <tweedledum>))
  (echo "gimbel")
  (call-next-method))
(defmethod (frumious (bandersnatch <tweedledoo>))
  (call-next-method)
  (echo "gyre"))
```

```lisp
=> (frumious (make <tweedledee>))
  slithy
  brillig
=> (frumious (make <tweedledoo>))
  slithy
  gimbel
  brillig
  gyre
=> (frumious (make <tweedledum>))
  gimbel
  brillig
=>
```

(b) (2 points) Is this output behavior uniquely determined by your definitions of <tweedledee>, <tweedledum>, and <tweedledoo> according to the Swindle/Scheme semantics? If not, how many others could there be? [No, 1]
“Wake up, Alice dear!” said her sister, “Why, what a long sleep you’ve had!”

“Oh, I’ve had such a curious dream!” said Alice, and she told her sister, as well as she could remember them, about all her strange adventures in Schemeland; and when she finished, her sister kissed her, and said “It was a curious dream, dear, certainly, but now run to your class; the final’s almost over!” So Alice got up and ran off, thinking while she ran, as well as she might, what a wonderful dream it had been AND HOW SHE NEEDS TO GET TO HOLLISTER 401 AND TAKE THE CS212 FINAL!

—THE END—