Fall, 1993 Name: ______________________
Grading TA's name: ______________________ Section Day and Time : __________

Com S 227 — Introduction to Computer Programming

Test on Chapter 7 (not including 7.5) (Version 1)

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

Reminders

You'll be able to do the problems quickly and correctly if you use the procedures and ideas of chapter 7. We will take off some points for inelegant solutions.

This test is closed book and closed 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.

Indentation is important to us for "clarity" points; if your code is sloppy, you will lose points.

Subset of Scheme You May Use to Define Other Procedures

Unless otherwise stated, when defining procedures you may only use: helping procedures that you define yourself, and the procedures and keywords that are included in the following list. (That is to say, you may *not* use Scheme procedures and keywords that are absent from the following list.) The notation `c...r` means `caar`, `cadr`, `cddr`, `cdar`, `caaar`, etc. You may only use `define` at the top level.

```
#t, #f, *, +, -, /, <, <=, =, >=, >,
abs, acos, add1, and, append, apply, asin, atan, atom?, atomic-item?, begin, boolean?,
car, cdr,  c...r, ceiling, cond, cons, cos, deep-recur-abs, define, display
else, eq?, equal?, eqv?, error, exp, expt, flat-recur, floor, for-each, if, integer?,
lambda, length, let, letrec, list, list-ref, log, map, max, min, modulo,
negative?, newline, not, null?, number?, or, pair?, positive?, procedure?,
quote, quotient, read, real?, remainder, reverse, round,
sin, sqrt,  string=?, string-append, string-ci=?, string-length, substring,
sub1, symbol?, symbol->string, tan, truncate, write, writeln, zero?
```

Types of Built-in Procedures from Chapter 7

Name	: Type
`map`	: (→ ((→ (S) T) (list S)) (list T))
`for-each`	: (→ ((→ (S) void) (list S)) void)
`apply`	: (→ ((→ (datum ...) T) (list datum)) T)
`flat-recur`	: (→ (T (→ (S T) T)) (→ ((list S)) T))

1. (10 points) Using the procedure `for-each`, define a procedure, `print-conversation`, that takes a list of strings, and prints the strings, one per line, surrounded by `‘‘` and `’’`. (Note that these are 2 characters each, not double quotes like ("').) For example, consider the following transcript

```
> (print-conversation '("Joe?" "Yes, April?" "Can you take out the trash?" "ok"))
‘‘Joe?’’
‘‘Yes, April?’’
‘‘Can you take out the trash?’’
‘‘ok’’
> (print-conversation '("April?" "Yes, Joe?" "I'm back!"))
‘‘April?’’
‘‘Yes, Joe?’’
‘‘I'm back!’’
>(print-conversation '())
>
```

You must use the procedure `for-each` *in your solution.*

2. (10 points) Using either the procedure `map` or `flat-recur`, define a procedure `turkey-talk` that takes a list of symbols, `ls`, and returns a list where each symbol is replaced by the symbol `gobble`. For example,

```
(turkey-talk '()) ==> ()
(turkey-talk '(hello turkey)) ==> (gobble gobble)
(turkey-talk '(we would like to have you for dinner))
          ==> (gobble gobble gobble gobble gobble gobble gobble gobble)
```

You must use either `map` *or* `flat-recur` *in your solution.*

3. (10 points) Write a procedure `extend-f-c` that takes a procedure, `f`, of type

$$(\rightarrow \text{(number number) number})$$

and returns a procedure that: takes two lists of numbers and returns a list of numbers which is the result of applying `f` to the corresponding elements of each list. You may assume that the two lists have the same length (but you don't have to assume that). For example,

```
((extend-f-c *) '() '()) ==> ()
((extend-f-c (lambda (x y) x)) '(3 4) '(5 6)) ==> (3 4)
((extend-f-c (lambda (x y) (/ (+ x y) 2))) '(4 4 6) '(50 60 70)) ==> (27 32 38)
((extend-f-c expt) '(2 2 2 2 2) '(1 2 3 4 5)) ==> (2 4 8 16 32)
```

4. (20 points) Define a procedure **`std-deviation`** that takes two (2) or more numbers and returns the standard deviation of the argument numbers. The standard deviation is defined as follows.

$$\texttt{(std-deviation } x_1 \ \ldots \ x_n\texttt{)} = \sqrt{\frac{(n \times (x_1^2 + \cdots + x_n^2)) - ((x_1 + \cdots + x_n)^2)}{n \times (n-1)}}$$

You can assume that **`std-deviation`** is called with at least two arguments. For example,

```
(std-deviation 1 1) ==> 0
(std-deviation -10 0 10) ==> 10
(std-deviation -20 -10 10 5.0 15 20) ==> 15.383974345619102
```

5. (20 points) Define a procedure `mins` that takes a (possibly empty) list of non-empty lists of numbers, `lls`, and returns a list of the minimums of each nested list. That is, the type of `mins` is

```
(-> ((list (non-empty-list number))) (list number))
```

For example,

```
(mins '()) ==> ()
(mins '((1 2) (3 4 5) (100 200 7 93) (44))) ==> (1 3 7 44)
(mins '((72 -12 -4 9) (1 2 -3 4 -5 6 -7))) ==> (-12 -7)
```