Fall, 1993 Name: _____

Grading TA's name: _____ Section Day and Time : _____

Com S 227X — Introduction to Computer Programming

Test on Chapters 1 and 2 (Version 1)

This test has 5 questions and pages numbered 1 through 4.

Reminders

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, *, +, -, /, =, and, begin, boolean?, car, cdr,  c...r, cond, cons,
define, else, eq?, equal?, eqv?, error, if, lambda, list, not, null?, number?,
or, pair?, procedure?, quote, symbol?, writeln
```

Directions from the "Course Specification"

You are encouraged to use helping procedures.

For programming problems, there is usually more than one right answer, hence it is important to write a clear solution. Answers to programming questions will be scored with the following breakdown of points:

- 60% for elegance and clarity (whether your program solves the given problem correctly in the given subset of the language)
- 40% for correctness (using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution)

You should try to eliminate all syntax errors from your programs, as they affect both the correctness and clarity of your solution; you may receive no points for programs with major syntax errors. Checking of external inputs, and other conventions for dealing with humans are not important (for this class), unless the problem says so.

1. (10 points) For this problem you can't use quoted lists or the procedure `list`.
   Using only `define`, `cons`, the empty list `'()`, and the following quoted symbols:

   ```
   'understand 'plan 'do 'examine
   ```

   write down a series of definitions followed by an expression that constructs the list

   ```
   (understand (plan) ((do) examine))
   ```

2. For both parts of this problem, do *not* use any abbreviations such as `caar` or `cddr`.
   Suppose we define the list `records` as follows.

   ```
   (define records '(cash (purchases (watch 40) (book 50))
                          (loans (student 20000) (personal 1000))))
   ```

   (a) (6 points) Write an expression using *only* the procedures `car` and `cdr` that returns the symbol `purchases` from `records`.

   (b) (8 points) Write an expression using *only* the procedures `car` and `cdr` that returns the number `20000` from `records`.

3. (4 points) Write the values returned by each of the following expressions.

   (a) `(symbol? (cdr (cons 'novel (cons 3 '())))`

   (b)
```
(cond
   ((eq? 'a 'b) 227)
   ((equal? '(born) (cons 'born '())) 93.4)
   (else (cons 32 (cons 0 (cons 273 '())))))
```

4. (10 points) Define a procedure `a-ize` that takes a list of symbols, `ls`, and returns a list of lists of the same length as `ls`, but such that in place of each symbol s in `ls`, the list `(a` s`)` appears. For example,

```
(a-ize '())  ==>  ()
(a-ize '(man plan canal))  ==>   ((a man) (a plan) (a canal))
(a-ize '(plan canal))  ==>  ((a plan) (a canal))
(a-ize '(canal))  ==>  ((a canal))
(a-ize '(one two one two three four))
           ==> ((a one) (a two) (a one) (a two) (a three) (a four))
```

5. (12 points) Define a procedure `translate-num` that takes a list of symbols, `los`, and returns a list that is just like `los` except that each occurrence of `un` is changed to `one`, and each `deux` is changed to `two`. All other symbols are put in the result unchanged. For example,

```
(translate-num '()) ==> ()
(translate-num '(un day at the un building deux wars stopped))
            ==> (one day at the one building two wars stopped)
(translate-num '(un deux deux un)) ==> (one two two one)
(translate-num '(deux deux un)) ==> (two two one)
```