Spring, 1993 Name: ____________________
Grading TA's name: ____________________ Section Name : ____________

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, boolean?, car, cdr,  c...r, cond, cons,
define, else, eq?, equal?, eqv?, error, if, lambda, list, not, null?, number?,
or, pair?, procedure?, quote, symbol?
```

Directions from "Course Policies and Procedures"

You are encouraged to use auxiliary (helping) functions whenever they are helpful.

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

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. Using only `define`, `cons`, the empty list `'()`, and the following quoted symbols:

   ```
   'wow 'what 'a 'list
   ```

   construct the list: `((wow) what (a list))`.

2. Suppose we define the list `poets` as follows.

   ```
   (define poets '(((Robert Lee Frost) (1874 1963))
                   ((Elizabeth Barrett Browning) (1806 1861))))
   ```

   (a) (6 points) Write an expression using only the procedures `car` and `cdr` that returns the symbol `Frost` from `poets`. For this problem you may *not* use any abbreviations such as `caar` or `cddr`.

   (b) (8 points) Write an expression using only the procedures `car` and `cdr` that returns the number 1861 from `poets`. For this problem you may *not* use any abbreviations such as `caar` or `cddr`.

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

   (a) `(null? (cdr (cons 'happy '())))`

   (b) `(equal? (cons 'go-lucky '()) (if #t (cons 'go-lucky '()) 'cheer))`

4. (12 points) Define a procedure `double` that takes a list, `ls`, and returns a list of twice as long such that each occurrence of an item in the original list is replaced by two occurrences in the result. For example,

```
(double '(1 2 3 4 5 6 7 8))  ==>  (1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8)
(double '(hughes milton dante))  ==>  (hughes hughes milton milton dante dante)
(double '())  ==>  ()
```

5. (12 points) Define a procedure `wow-second` that takes a symbol, `sym`, and a list of symbols, `los`, and returns a list that is just like `los` except that the second occurrence of `sym` (if any) is replaced by the symbol `wow` For example,

```
(wow-second 'great '(this great great car))  ==>  (this great wow car)
(wow-second 'great '())  ==>  ()
(wow-second 'awesome '(man it was awesome))  ==> (man it was awesome)
(wow-second 'super '(super just super super super))  ==> (super just wow super super)
(wow-second 'super '(super just))  ==> (super just)
(wow-second 'super '(super just really super and super)) ==> (super just really wow and super)
```