Fall, 1992 Name: _____

Grading TA's name: _____ Section Letter : _____ Section Time : _____

CS 227X — Introduction to Computer Programming

Test on Chapters 1 and 2 (Version 1)

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

Special Directions for This Test

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.

The "evidence" (argument) that your code is correct described in the *Red Tape* is not needed for this test.

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.

```
#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 the Course "Red Tape"

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. (4 points) Without using quoted lists, construct the list `(bb (ccc dddd))`. For this problem you may only use `cons`, quoted symbols, and the empty list.

2. (5 points) Suppose we define the list `caged-magician` as follows.

   ```
   (define caged-magician '(cage (chain (Harry Houdini) chain-lock) cage-lock))
   ```

   Write an expression using only the procedures `car` and `cdr` that returns the symbol `Houdini` from `caged-magician`. For this problem you may *not* use any abbreviations such as `caar` or `cddr`.

3. (8 points) Define a procedure `seconds-of-all` that takes a list of lists, `ll`, and returns a list of the second top-level item in each of the sublists of `ll`. You may assume that each sublist in `ll` has at least 2 elements. For example,

```
(seconds-of-all '((1 2)  (a b)  (e f h)))  ==>  (2 b f)
(seconds-of-all '(((a b) e)  ((lots of) (fun eh))))  ==>  (e (fun eh))
(seconds-of-all '())  ==>  ()
```

4. (12 points) Define a procedure `pattern-3-move` that takes a list of 3 items, `lst`, and another list of 3 items, `pattern`, and returns a list of the 3 items from `lst` arranged according to the pattern given in `pattern`. The `pattern` argument contains the numbers 1, 2, and 3 in some order. If 1 is second in `pattern`, then the first item of `lst` should appear second in the result. If 1 is third in `pattern`, then the first item of `lst` should appear third in the result. If 2 is first in `pattern`, then the second item of `lst` should appear first in the result, and so on. For example,

```
(pattern-3-move '(a b c) '(3 2 1) )  ==>  (c b a)
(pattern-3-move '(a b c) '(1 3 2) )  ==>  (a c b)
(pattern-3-move '(Mozart Hydan Bach) '(1 2 3))  ==> (Mozart Hydan Bach)
(pattern-3-move '(Mozart Hydan Bach) '(2 3 1))  ==> (Hydan Bach Mozart)
```

You may assume that the arguments have the right form.

5. (10 points) Define a procedure `precedes` that takes 3 arguments: symbols `prior` and `latter`, and a list of symbols, `los`. The procedure returns a boolean. The result is `#t` if and only if there is no occurrence of `latter` before the first occurrence of `prior` in `los`. That is, the result is `#f` when `latter` occurs in `los` before the first occurrence of `prior`. For example,

```
(precedes 'a 'b '(a c b))  ==>  #t
(precedes 'a 'b '(e a c))  ==>  #t
(precedes 'a 'b '(b a c b))  ==>  #f
(precedes 'dos 'tres '())  ==>  #t
(precedes 'dos 'tres '(eins zwei drei vier))  ==>  #t
(precedes 'dos 'tres '(eins zwei dos dos funf))  ==>  #t
(precedes 'dos 'tres '(eins tres zwei dos dos funf))  ==>  #f
```