Fall, 1992 Name: _____

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

Com S 227X — Introduction to Computer Programming

Test on Chapters 3 and 4 (Version 1)

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

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, *, +, -, /, <, <=, =, >=, >,
abs, acos, add1, and, append, asin, atan, atom?, begin, boolean?,
car, cdr,  c...r, ceiling, cond, cons, cos, define,
else, eq?, equal?, eqv?, error, exp, expt, floor, if, integer?,
lambda, length, list, list-ref, log, max, min, modulo,
negative?, not, null?, number?, or, pair?, positive?, procedure?,
quote, quotient, real?, remainder, reverse, round, sin, sqrt, sub1, symbol?,
tan, truncate, writeln, zero?
```

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) Define a procedure `sum-to-n` that takes an integer, `n`, and returns $1 + 2 + \cdots + n$. You may assume that `n` is not negative. For example,

```
(sum-to-n 0) ==> 0
(sum-to-n 1) ==> 1
(sum-to-n 3) ==> 6
(sum-to-n 6) ==> 21
```

2. (6 points) Define a procedure `cdr-n` that takes an integer `n` and a list `ls` and returns a list that is the result of applying `cdr` to `ls` `n` times. For example,

```
(cdr-n 0 '(a b c)) ==> (a b c)
(cdr-n 2 '(c s 2 2 7 X)) ==> (2 2 7 X)
(cdr-n 3 '(a b c)) ==> ()
```

   The procedure `cdr-n` must give an error message when the list `ls` is not big enough. For example, the call `(cdr-n 4 '(a b c))` prints the message `Error: Index too high for list` and has no specified return value.

3. (14 points) Define a procedure `add-odd-all` that takes a tree of integers, and returns the sum of the odd integers in the tree. (Don't ask us what a "tree" is, you're supposed to know that.) For example,

```
(add-odd-all '()) ==> 0
(add-odd-all '(1 2 4 3 5 8 16)) ==> 9
(add-odd-all '((3 4) 1 2 ((8 9) 7))) ==> 20
```

4. (16 points) Define an iterative procedure `list-index` that takes a list `ls` and a datum `item`. If `item` occurs as a top-level item in `ls`, then `(list-index ls item)` returns the zero-based position of the first top-level occurrence of `item`, otherwise `list-index`, returns -1. For example,

```
(list-index '() 'a) ==> -1
(list-index '(a b c a) 'f) ==> -1
(list-index '(a b c a) 'a) ==> 0
(list-index '((fie) (foo) (fo) (fa) (foo)) '(fa)) ==> 3
```

   Only solutions that are iterative will get points. (Don't ask us what "iterative" or "top-level" mean, you're supposed to know that.)

5. (20 points) Define a procedure `batch-xerox-all` that takes a tree and returns a tree that is the same as the parameter, except that it has all 3-element lists of the form `(xerox` n d`)` replaced by lists of the form (*d* ... *d*), with n copies of d. You may assume that each list that starts with the symbol `xerox` has 3 elements of this form and that in such lists n is non-negative. For example,

```
(batch-xerox-all '())  ==>  ()
(batch-xerox-all '(xerox 5 hi))  ==>  (hi hi hi hi hi)
(batch-xerox-all '(xerox 0 hi))  ==>  ()
(batch-xerox-all '(((xerox 2 foam) a (3)) (xerox 1 x)))
             ==>  (((foam foam) a (3)) (x))
```

You may assume that the arguments have the right form.