Spring, 1993 Name: _____

Grading TA's name: _____ Section Name : _____

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.

Reminders

This test is closed book and closed notes.

As on homeworks, for helping procedures, you must write TYPE, ENSURES, (and REQUIRES if appropriate) comments.

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, asin, atan, atom?, atomic-item?, begin, boolean?,
car, cdr,  c...r, ceiling, cond, cons, cos, define, denr
else, eq?, equal?, eqv?, error, exp, expt, floor, if, integer?,
lambda, length, list, list-ref, log, make-ratl, max, min, modulo,
negative?, not, null?, numr, number?, or, pair?, positive?, procedure?,
quote, quotient, real?, remainder, reverse, round, sin, sqrt, sub1, symbol?,
tan, truncate, writeln, zero?
```

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. (8 points) Define a procedure `prod-of-cubes` that takes an integer, `n`, and returns $1^3 \times 2^3 \times \cdots \times n^3$. If `n` is 0, return 1. You may assume that `n` is not negative. For example,

```
(prod-of-cubes 0) ==> 1
(prod-of-cubes 1) ==> 1
(prod-of-cubes 2) ==> 8
(prod-of-cubes 3) ==> 216
(prod-of-cubes 9) ==> 47784725839872000
```

2. (8 points) Define a procedure `first-n` that takes an integer `n` and a list `ls` and returns the first `n` items in the list ls. For example,

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

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

3. (15 points) This is a problem about rational numbers. Recall that the basic procedures for rational numbers are as follows.

`make-ratl`	: integer, integer → rational
`numr`	: rational → integer
`denr`	: rational → integer

(Don't ask us what these do, you're supposed to know that.) If you need other procedures on rationals, you'll have to write them yourself (that is part of this problem).

Write a procedure `radd1` that takes a rational number, and returns a rational that is the sum of that rational plus 1/1. Recall that $(a/b) + (c/d) = (ad + cb)/bd$. For example, using the `rprint` defined in the book to show answers:

```
(rprint (radd1 (make-ratl 2 3))) =prints=> 5/3
(rprint (radd1 (make-ratl 1 2))) =prints=> 3/2
(rprint (radd1 (make-ratl 17 293))) =prints=> 310/293
(rprint (radd1 (make-ratl -17 3))) =prints=> -14/3
```

You must use the basic procedures for rationals correctly in your solution. Your code should be independent of the representation of rationals; that is, low level solutions will lose points. (Don't ask us what "representation independent" or "low level" means, you're supposed to know that.)

4. (16 points) Define a procedure `decision-tree` that takes a symbol, `sym`, and a tree of symbols, `tos`, and returns a tree of booleans. The tree returned has the same structure as `tos`, but has a `#t` in each place where there is a symbol in `tos` that is the same as `sym` and a `#f` in each place where `tos` has a symbol that is not the same as `sym`. (Don't ask us what a "tree" is, you're supposed to know that.) For example,

```
(decision-tree 'x '()) ==> ()
(decision-tree 'x '(() ())) ==> (() ())
(decision-tree 'foo '(on foo the bar foo)) ==> (#f #t #f #f #t)
(decision-tree 'flo '((noble flo smith) () (hi) (data flo (flo))))
               ==>  ((#f #t #f) () (#f) (#f #t (#t))))
```

5. (16 points) This is a problem about iteration. Define an *iterative* procedure `minimum` that takes a non-empty list of numbers and returns a number from the list that is at least as small as any other number in the list. For example,

```
(minimum '(3 2 4)) ==> 2
(minimum '(-5.3 -2 4 9 18 1)) ==> -5.3
```

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