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

Com S 227X — Introduction to Computer Programming

Test on Chapters 5 and 6 (Version 1)

This test has 7 questions and pages numbered 1 through 7.

Reminders

This test is closed book and closed notes. No calculators allowed.

As on homeworks, for helping procedures, you must write TYPE, ENSURES or EFFECT, (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.

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, degree, display
else, eq?, equal?, eqv?, error, exp, expt, floor, if, integer?,
lambda, leading-coef, length, let, letrec, list, list-ref, log, max, min, modulo,
negative?, newline, not, null?, number?, or, pair?, poly-cons, positive?, procedure?,
quote, quotient, read, real?, remainder, rest-of-poly, reverse, round,
sin, sqrt,  string=?, string-append, string-ci=?, string-length, substring,
sub1, symbol?, symbol->string, tan, the-zero-poly, truncate, write, 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. (5 points) Convert the following binary number to decimal (by hand): 1110010010

2. (5 points) Convert the following decimal number to binary (by hand): 19

3. (5 points) Consider the following procedure definitions.

```
(define sum-to-n
  (lambda (n)
    (sum-to-n-it n 0)))

(define sum-to-n-it
  (lambda (n acc)
    (if (zero? n)
        acc
        (sum-to-n-it (sub1 n) (+ n acc)))))
```

Rewrite the definition of `sum-to-n`, so that `sum-to-n-it` is not defined as a separate procedure, but is defined using a `letrec`.

4. (10 points) In the following expression:

   (a) Draw a squiggly arrow ($\rightsquigarrow$) going from each parameter (variable bound in a `let`) to the value to which it is bound.

   (b) Draw a straight arrow ($\rightarrow$) from each use of a variable in an expression to its associated introduction as a parameter in a `let`.

   (c) Give the output of the expression.

```
(let ((x   1)     (y   2))

  (let ((x   3)     (z   x))

    (writeln "x is: "  x  " y is: "  y  " and z is: "  z)))
```

You may assume that `writeln` has already been defined.

5. (10 points) In the following expression:

   (a) Draw a squiggly arrow ($\rightsquigarrow$) going from each parameter (variable bound in a `lambda`, `let`, or `letrec`) to the value to which it is bound.

   (b) Draw a straight arrow ($\rightarrow$) from each use of a variable in an expression to its associated introduction as a parameter in a `let`.

   (c) Give the output of the expression.

```
(let ((p   (lambda (x)

             (display (+ 2 x))) ))

  (letrec ((fe  (lambda (ls)

                  (if (not (null? ls)) (begin

                                         (p (car ls))

                                         (fe (cdr ls))

                                         (newline)

                                         (p (car ls))))) ))

    (fe '(3 4 5))))
```

6. (20 points) This is a problem about polynomials. Recall that the 5 basic definitions for polynomials and their types are:

```
the-zero-poly: poly
degree: poly -> integer
leading-coef: poly -> real
rest-of-poly: poly -> poly
poly-cons: integer, real, poly -> poly
```

Using the 5 basic definitions and any helping procedures you define yourself, define a procedure `even-degree-terms` that takes a polynominal and returns the polynomial which contains the terms of even degree of the original. For example, passing the polynomial

$$2x^4 + 8x^3 + 6x + 7.2$$

to `even-degree-terms` results in the following polynomial.

$$2x^4 + 7.2$$

Passing the zero polynomial to `even-degree-terms` results in the zero polynomial, since $0x^0$ has even degree.

Low level solutions will lose points. (Don't ask us what the 5 basic definitions for polynomials do, you're supposed to know them.) If you want to use other procedures on polynomials besides the 5 basic definitions, you will have to write their definitions here.

7. (20 points) Define an interactive procedure, **`draw-rectangles`**, that takes no arguments and has the following behavior.

   The procedure **`draw-rectangles`** prompts for a width and height, and then prints a rectangle of asterisks of the given width and height. It continues prompting and printing in this way until a width of zero is entered. When a width of zero is entered, **`draw-rectangles`** writes `Bye.` instead of continuing.

   You no *not* have to check that the input is valid, and may assume that the width and height entered are always non-negative integers. For example,

```
> (draw-rectangles)
Width? (or 0 to exit) 7
Height? 4
*******
*******
*******
*******
Width? (or 0 to exit) 4
Height? 1
****
Width? (or 0 to exit) 0
Bye.
>
```