Spring, 1992 Name: _____

CS 342 — Principles of Programming Languages

Final Exam (Version 2)

Before you begin, please take a moment to look over the entire test so that you can budget your time.

Short answer questions will be graded on the basis of completeness (did you list all the relevant parts of the answer?) soundness (did you make any errors of fact or contradict yourself?) and clarity. You can help ensure clarity by giving examples where possible.

For essays, critical justifications, and the like there may be no one right answer; so the criteria for judgement will be whether your essay is:

- clear (you use examples where appropriate, you write clear sentences, diagrams, and you avoid excess verbosity),
- sound (you start from facts or reasonable assumptions, your logic is convincing), and
- complete (you consider relevant aspects of the problem and alternatives).

The highest grade will be earned by an answer that is, in addition, concise and elegant; that is, answers that say the most with the least amount of excess. Extra consideration will be given to answers that are especially creative. For programming problems, there is usually more than one right answer, hence it is important to present with your program evidence that it is correct. Answers to programming questions will be scored on the following points:

- elegance and clarity (conciseness, using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution, and how clear and convincing your argument is that the code is correct)
- correctness (whether your program solves the given problem correctly in the given programming language)

You should try to eliminate all syntax errors from your programs, as they will affect both the correctness and clarity of your solution; you may receive no points for programs with major syntax errors. Run-time space and time efficiency of programs is not important, unless the problem says so. Also, checking of external inputs, prompting, labeling output, and other conventions for dealing with humans are not important (for this class), unless the problem says so.

The breakdown of points between these categories is roughly as follows.

- 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 given language)

You are encouraged to use auxiliary subroutines whenever they are helpful.

For this test you may use without further mention any code in the book from chapter 1, LISP, Scheme, CLU, Smalltalk, or Prolog. However, make sure that:

- When solving a problem in language X, the code you are using is written in the book in language X or was translated from an earlier chapter into language X in the library for langauge X. Functions that you may assume are available in every language (except Prolog) include: `+1`, `or`, `and`, `not`, `<>`, `<=`, `divides`, `mod`, `gcd`, and `abs`.

  That is, don't use code from language Y when solving a problem in language X.

- If you have any doubts about our being able to find the code you are using, you give us a page number where we can find the code. This is especially important if the chapter has multiple versions of a function with different names.

1 Scheme

1. (5 points) Consider a variant of Scheme with dynamic scoping. Would the interpreter for this variant of Scheme need to make a closure for a `lambda` expression? Briefly explain your answer.

2. (5 points) Consider the following Scheme function.

```
(set cn
  (lambda (n)
    (lambda (f)
      (lambda (a)
        (if (= n 0)
            a
            (f (((cn (- n 1)) f) a)))))))
```

What is the result, if any, of the following Scheme expression?

```
(((cn 2) (lambda (x) (cons (* 4 (car x)) x))) '(9))
```

(Show your work if you wish partial credit.)

3. (8 points) Write, in Scheme, a function `right-section` such that

`((right-section f x) y)` = `(f y x)`.

For example, `((right-section - 1) 7)` is 6, and `((right-section * 2) 5)` is 10.

2 General Problem Solving

In this section of the test, you will be directed to solve a problem, and will be required to choose a paradigm and language to solve the problem in, defend your choice, and then solve the problem. The languages you may use are the textbook's: LISP, Scheme, CLU, Smalltalk, and Prolog.
Part of each problem may be to refine the problem statement. Scoring will be one-quarter (1/4) on the choice and defense of paradigm and language (graded as an essay question) and three-quarters (3/4) on the solution itself. Your defense of your choice of paradigm and language should consider the alternatives.

1. (20 points) In this problem you will code a function or operation or relation, `predict-weather` that predicts what the weather will be tomorrow. The input to `predict-weather` is a list of facts about the current weather conditions. For example, in LISP such a list might be written,

   ```
   '((sky cloudy) (temperature cold) (barometer falling))
   ```

   In Prolog you would use a list such as

   ```
   (cons (pair sky cloudy) (cons (pair temperature cold)
         (cons (pair barometer falling) nil)))
   ```

   In Smalltalk you could use a Dictionary to represent such a list. If you wish you may assume (by writing down your assumption) that the elements of the list always occur in the order given above.

   The output of `predict-weather` is one of the following symbols: `raining`, `snowing`, `sunny`, or `cloudy`. The prediction given by `predict-weather` should obey the the following rules:

   - The prediction is `snowing`, if the sky is `cloudy` and the temperature is `cold` and the barometer is `falling`,
   - The prediction is `raining`, if the sky is `cloudy` and the temperature is `hot` or `warm` and the barometer is `falling`,
   - The prediction is `cloudy`, if the sky is `cloudy` and the barometer is `stable`,
   - The prediction is `sunny` if the barometer is `rising`,

2. (25 points) In trying to break a secret code, it is convenient to look at how many times a letter occurs in a piece of coded text. We will consider a text to be a list of numbers (each number representing a "letter"). In this problem you are to write a function or relation or operation `frequencies` that returns a list of 2-element lists (called "pairs"), of numbers and how many times that number occurs. The list of pairs does not have to be ordered in any particular way. For example, in LISP or Scheme,

```
-> (frequencies '(7 5 10 7 10 8 7))
((5 1) (10 2) (8 1) (7 3))
```

For example, in Prolog,

```
-> (infer? (frequencies (cons 9 (cons 8 (cons 8 nil))) Fs) (print Fs))
(cons (cons 9 (cons 1 nil)) (cons (cons 8 (cons 2)) nil))
-> (infer? (frequencies nil Fs) (print Fs))
nil
```

In Smalltalk or CLU you would also use lists as the input and output.

Remember to (a) write down your choice of paradigm and language, and (b) defend that choice against any alternatives.