COMP 210, Spring 2001, Homework 5
Due Wednesday, February 21, 2001 at the start of class

Before you start the homework, you should remind yourself of our General Advice, Advice on Homeworks, and Grading Guidelines. All are available from the class web site (http://www.owlnet.rice.edu/~comp210).

For this assignment, you should follow all the steps of the design methodology and include the results of each step as comments or code in the final materials that you submit.

1. (4 pts) In class, we discussed two data definitions for family trees (reproduced below). Some programs are easier to write using one data organization rather than the other. To explore this, *consider* what it would take to develop the following program using each of the two definitions.

   The program **`find-siblings`** consumes a list of family trees (either an **`FT`** or a **`person`**) and a symbol and produces a list of symbols. The symbols in the output list are the names of the siblings of the person named in the input symbol. You may assume that a name appears at most once in a family tree.

   a. (1 pt) Under which data definition is **`find-siblings`** easier to develop.

   b. (3 pts) Develop the program for the data definition that you named in part (a.) Be sure to show all the steps in the design methodology.

   This problem uses a list of family trees so that we may have multiple entry points into each family. For example, in the descendant tree on page 210 of the text, the list would contain the parent structures for Carl, Bettina, and Fred. As a reminder, here are the two data definitions for family trees:

   Child-centric family trees

```
;; An FT (for family tree) is either
   ;;     -  empty, or
   ;;     - (make-child name father mother year eyes)
   ;; where name and eyes are symbols,
   ;;       father and mother are FT's, and
   ;;       year is a number
   (define-struct child (name father mother year eyes))
```

   Parent-centric trees

```
;;  A person is a structure
;;     (make-person name year eyes children)
;;  where name and eyes are symbols, year is a num,
;;            and children is a list-of-person.

(define-struct person (name year eye-color children))

;; A list-of-person is either
;;  - empty, or
;;  - (cons f r)
;;  where f is a person and r is a list-of-person
```

The text discusses both types of trees in Sectios 14 and 15, respectively. It uses the name **`parent`** instead of **`person`** in the **`struct`** for parent-centric trees.

2. (3 points) Using the definitions for directory trees given in lecture 12 (online), develop the following program.

   `dup-names: directory → list-of-symbols` Your program should consume a directory tree and return a list containing all the names that occur more than once in the entire directory tree. (The function **`dup-names`** differs from the function **`any-duplicate-names?`** in the lab tutorial because it must check the entire directory tree. That is, if **`'foo`** occurs in two different subtrees of the directory, your program should find that duplication and report it by including **`'foo`** in the resulting list.)

   Extra credit : (1 point) Write **`dup-names`** so that any symbol that appears *more than twice* in the input appears *only once* in the output.

3. (4 points) Work Exercise 17.6.6 from the book (page 246 in my copy) to develop the program **DNAprefix**. Show all the steps in the design methodology.

4. Extra credit: (10 points) The fibonacci function *fib* is defined the equations

   $fib(0) = 1$

   $fib(1) = 1$

   $fib(n+2) = fib(n+1) + fib(n)$ for $n \geq 0$

   In the naïve implementation of this definition as a program, computing *fib*(*n*) requires O(2^*n*) (a number proportional to 2^*n*) steps (count each reduction performed by the DrScheme stepper as a step). It is not difficult to write a program that reduces the running time to O(*n*) steps. But it is possible to do much better. The function *fib*(*n*) can be computed in O(*log n*) steps. Write a program to compute *fib*(*n*) in O(*log n*) steps.

   Hints:

   1) Devise a recurrence expressing *fib(2*n)* and *fib*(*2*n*+1) in terms of *fib*(*n*) and *fib*(*n*+1).
   2) Initially design your program only for values of *n* that are powers of 2. Then revise it to work for all non-negative *n*.