COMP 210, Spring 2002, Homework 5
Due Wednesday, February 20, 2002 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 a ftn or a parent) 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.

   Note: this problem uses a **list of ftn** or a **list of parent** so that it has access to multiple entries into a single family.

   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.

Child-centric family trees

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

Parent-centric trees

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

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

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