COMP 210, Spring 2001, Homework 7
Due Wednesday, March 14, 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).

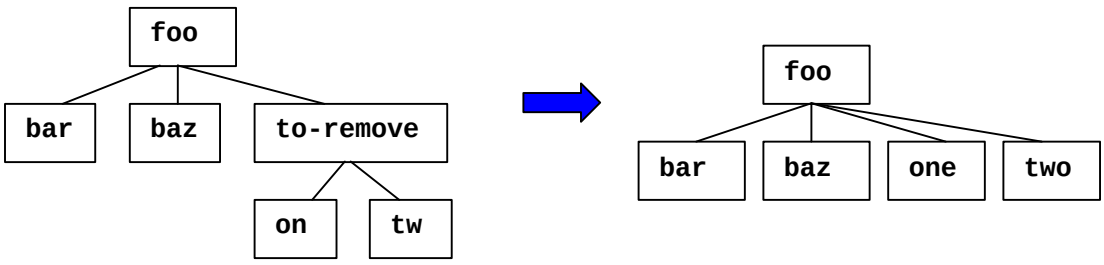
1. (7 Pts) The following data definition of directory trees was given in class:

```
; A file is a symbol (its name).
; A directory is a structure
;   (make-dir name contents)
;   where name is a symbol, and contents is a l-o-f-d.
; A list-of-files-and-directories (l-o-f-d) is one of
; - empty
; - (cons f lofd)
;   where f is a file, and lofd is a l-o-f-d
; - (cons d lofd)
;   where d is a directory, and lofd is a l-o-f-d.
```

a) (2 pts) Develop a function

```
; flatten-dir-once: directory symbol ->; (directory or l-o-f-d)
; Purpose: Returns a structure like the original directory, except
; that the named directory is removed, with its contents moved up
; one level.  If the specified directory name occurs more than once,
; only the first occurrence of the named directory is replaced
```

Here is a pictorial example in which the directory named `'to-remove` is removed:



You may assume that the directory contains at most one directory that matches the specified name.

Some more examples:

```
  (flatten-dir-once
    (make-dir 'cork (list 'comp210 'comp212 'comp311 'comp511))
    'cork)
= (list 'comp210 'comp212 'comp311 'comp511)
```

```
  (flatten-dir-once
    (make-dir 'cork
              (list 'comp210
                    (make-dir 'comp212
                              (list 'exam1 'exam2))
                    'comp311
                    'comp511))
    'comp212)

= (make-dir 'cork
            (list 'comp210
                  'exam1
                  'exam2))
                  'comp311
                  'comp511))
```

Hint: **`local`** is useful in the definition of the function that processes an **`l-o-f-d`**

b) (2 pts) Develop a function

```
; any-shadowed-names?: directory -> boolean
; Purpose: determines whether any directory directly or indirectly
; contains another directory or file of the same name.  It does NOT
; check for duplicated names in separate branches of the tree.  Note
; that this is simply the descendant relationship on trees.
```

c) (3 pts) Develop a function:

```
; sort-dir: directory -> list-of-symbol
; Purpose: returns a list of the names in the directory in ascending
; alphabetical order.
```

Hint: flatten the tree to a list of symbols and sort it. Use the following function **`symbol<=?`** to determine if one symbol precedes another in alphabetical order

```
(define (symbol<=? sym1 sym2)
  (string<=? (symbol->string sym1) (symbol->string sym2)))
```

2. (1 pt) Given the following definition for a point

```
;; a point is
;;    (make-point x y)
;; where x and y are numbers
(define-struct point (x y)
```

Use the Scheme functions **filter** and **sqrt** to create a function **within-1** that consumes a list of point and produces a list of point. The list that it produces should contain exactly those points that are within a distance of 1 from the point (0,0).

[That is, the square root of $(x-0)^2 + (y-0)^2$ is less than or equal to one.]

Use **local** to hide any helper functions that you write.

3. (2 pts) You are to build a simple model of how the Social Security Administration (SSA) might project the cost of benefits. Assume that each taxpayer has a record

```
;; a taxpayer is a
;;    (make-taxpayer  name age)
;; where name is a symbol and age is a number
(define-struct taxpayer (name age))
```

   Of course, the SSA keeps a list of all the taxpayers who are involved in the Social Security System. Once a year, they need to increment the age of each taxpayer and count the number of taxpayers old enough to receive benefits.

   Write two functions:

   a) Write **eligible: list-of-taxpayer -> list-of-symbol**. Your **eligible** function should return a list of the names of each taxpayer who is over 65.

   b) Write **make-older: list-of-taxpayer -> list-of-taxpayer**. Your **make-older** consumes a list of taxpayers and returns a list in which each taxpayers age has been increased by one. The SSA runs this at the beginning of each year.

   You should use Scheme's abstract functions (e.g, **filter**, **map**) where possible in your implementation. Use **`lambda`** for any helper functions that you pass into the abstract functions.

4. (Extra Credit: 4pts) Write a solution to the extra credit version of the **`dup-names`** problem (# 3) from Assignment 5 that runs in O(n log n) time on average where n is the number of symbols in the directory tree. The extra credit version of the problem stiputated that no name should appear in the result list more than once.

   Your solution must be purely functional. Worst case behavior may be O(n^2).