DUE: Thursday October 21, 2004, 11:59 p.m.

All of the functions that you write for this homework should be defined in the same source file, hw3.scm. Questions 4-5 are each worth about twice as much as each of Questions 1-3. Use helper functions where appropriate to simplify your code.

As before, there are three steps to turning in your homework:
a.) Use the turnin link from the calendar page; follow the instructions and submit the file hw3.scm.
b.) Then complete the web survey that you are directed to.
c**.) Print a copy of your hw3.scm and turn it in during class on Friday October 22. Write your name and UW Net ID prominently on the first page.**

1. Write a procedure `(squares-up-to n)` that takes one argument, a positive integer, and returns a list of n values from 1^2 to n^2.

2. Write a procedure `(build-summer p)` that takes a procedure `p` and returns a procedure that:
   - takes one argument, a list `m`,
   - applies procedure p to every number in the list m,
   - and returns the sum of the result of applying procedure p to every number in the list.

   Example: ((build-summer cube) (list 1 2 3)) → (1 + 8 + 27) = 36
   You need not necessarily execute the steps in the order given above, but your function must produce the same result as if you had.

3. Write a procedure `(delete k m)` that takes a single number `k` and a list of numbers `m` and returns a list with all occurrences of `k` deleted.

4. A mobile consists of one or more branches. Each branch is a rod of length 0 or more from which hangs either a weight or another mobile. Each branch is attached to the root of the mobile at an angle from 0 to 2π around the vertical axis. For instance, mobile m1 (see figure later) has two branches, one at an angle of 0 degrees and one at 180 degrees – from both branches hangs a single weight. Mobile m3 also has two branches, but from each branch hangs another mobile. We can represent a mobile with angles and branches using lists as follows:

   The procedure new-mobile takes an argument list with an even number of entries. The first entry is an angle, the second entry is a branch structure. There can be 1 or more pairs of arguments. The resulting data structure is a list with two entries. The first entry is a list of angles, the second entry is a list of branch structures. Take a moment to understand how the following procedure works:

```
(define (new-mobile . v)
  (define (pick-list get-item get-next shrink)
    (if (null? shrink)
      '()
      (cons (get-item shrink) (pick-list get-item get-next (get-next shrink)))))
  (list (pick-list car cddr v) (pick-list cadr cddr v)))
```

(continued)

A branch is constructed from a length (which must be a number greater than or equal to 0) together with a structure, which may be either a number (representing a simple weight) or another mobile:

```
(define (new-branch length structure)
  (list length structure))
```

a. Write procedures (angle-list m) and (branch-list m) that return the list of angles and the list of branch structures from a mobile m. This is easy if you understand the data structure.

b. Write procedures (length b) and (structure b) that return the length value and the structure value (either a weight or a mobile) from a branch b.

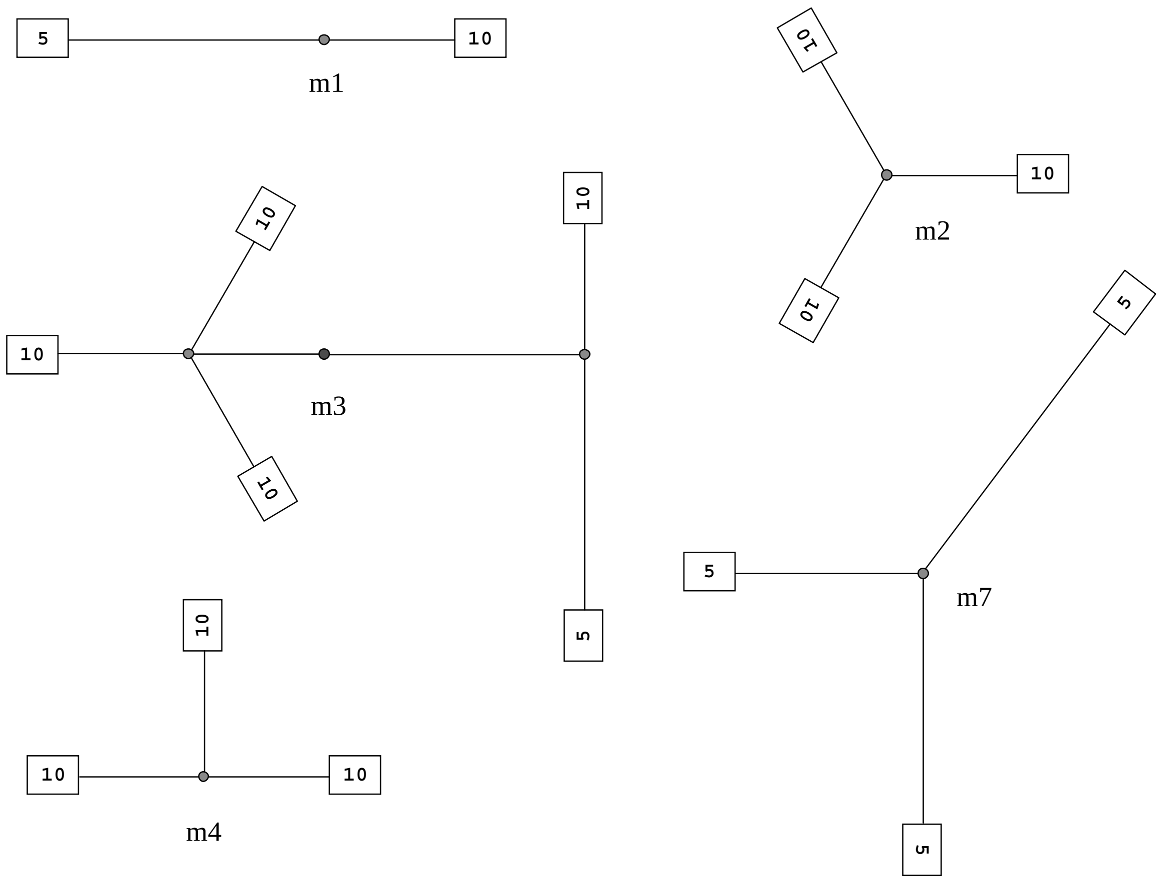
c. Write procedures `(branch-weight b)` and `(mobile-weight m)` that return the weight of a branch b or a mobile `m`. Note that these two procedures call each other as needed, depending on the structure of the mobile.

   Details: the weight of a mobile is the sum of the weights of its branches. For a branch, it depends on whether the structure is a plain weight or a mobile (how can you tell?). For a plain weight, then the weight is just the weight given. If it is a mobile, then the weight of the branch is the total weight of the mobile.

5. A mobile is said to be *balanced* if the torque applied by all its branches sums to zero in all directions. This is true if (1) the sum over all attached branches of [rod length multiplied by the total weight hanging from that rod multiplied by sin(attachment angle)] is very close to zero, *and* (2) the sum over all attached branches of [rod length multiplied by the total weight hanging from that rod multiplied by cos(attachment angle)] is very close to zero, *and* (3) each of the mobiles hanging off its branches is balanced. A branch with a plain weight is always balanced. A branch with a mobile is balanced if that mobile is balanced.

   Write a predicate `(mobile-balanced? m)` that tests whether a mobile `m` is balanced. It will be helpful to write a helper function `branch-balanced?` that interacts with `mobile-balanced?` to determine the balance of a branch.

These show some of the mobiles from hw3-test-it.scm, viewed from above:



Output of hw3-test-it.scm

```
Welcome to DrScheme, version 208.
Language: Standard (R5RS).
squares-up-to
(squares-up-to 1) => (1) : (1)
(squares-up-to 5) => (1 4 9 16 25) : (1 4 9 16 25)
build-summer
((build-summer cube) (list 0 1 2 3)) => 36 : 36
((build-summer (lambda (x) (expt 2 x))) (list 0 1)) => 3 : 3
((build-summer identity) (list 1 2 3 4 5 6 7)) => 28 : 28
delete
(delete 1 (list 1)) => () : ()
(delete 1 (list 1 1 1)) => () : ()
(delete 1 (list 1 2 3)) => (2 3) : (2 3)
(delete 1 (list 2 3 4)) => (2 3 4) : (2 3 4)
(delete 1 (list 3 2 1)) => (3 2) : (3 2)
mobiles
(branch-weight b1) => 10 : 10
(branch-weight b2) => 5 : 5
(mobile-weight m1) => 15 : 15
(mobile-weight m2) => 30 : 30
(branch-weight b2.1) => 15 : 15
(branch-weight b2.2) => 30 : 30
(mobile-weight m3) => 45 : 45
(branch-balanced? b1) => #t : #t
(branch-balanced? b2) => #t : #t
(branch-balanced? b3) => #t : #t
(branch-balanced? b4) => #t : #t
(branch-balanced? b5) => #t : #t
(mobile-balanced? m1) => #t : #t
(mobile-balanced? m2) => #t : #t
(branch-balanced? b2.1) => #t : #t
(branch-balanced? b2.2) => #t : #t
(mobile-balanced? m3) => #t : #t
(mobile-balanced? m4) => #f : #f
(mobile-balanced? m5) => #f : #f
(branch-balanced? b2.3) => #t : #t
(branch-balanced? b2.4) => #f : #f
(branch-balanced? b2.5) => #f : #f
(mobile-balanced? m6) => #f : #f
(mobile-balanced? m7) => #t : #t
>
```