All of the functions that you write for this homework should be defined in the same source file, hw3.scm. Each question is worth 2 points, for a total of 20.

Note that some of these questions use input/output operations that are only available when the language selected is "MzScheme". Therefore you should switch the language level from R5RS to MzScheme.

1. In the Lists lecture I gave an example of building rational numbers using lists. The make-rat constructor can be rewritten to divide out the gcd of the numerator and the denominator, thereby reducing them to lowest terms. It can be further enhanced to normalize the sign of the two numbers so that if the rational number is positive, both the numerator and denominator are positive, and if the rational number is negative, only the numerator is negative. Write the constructor `(make-rat d n)` that implements these enhancements and the two related accessor functions `(numer x)` and `(denom y)`. The rat-operators are provided to you in a separate file.

2. Each procedure in question 2 returns a list of integers.

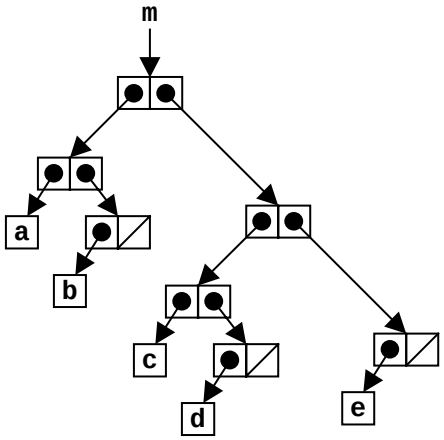
a. Write a procedure `(count-down-from n)` that takes one argument, a positive integer, and returns a list of the positive numbers from n down to 1.

b. Write a procedure `(count-up-to n)` that takes one argument, a positive integer, and returns a list from 1 up to the argument.

3. Write a procedure `(add-transformed m p)` that takes a list of numbers `m` and a procedure `p` and returns the sum of the results of applying the procedure `p` to each number in the list `m`.

4. 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.

5. Questions 5 and 6 refer to this data structure:



a. Show how you would construct the data structure using a set of (cons …) operations. Define variable m-5a to hold the result.

b. Can you construct this data structure using nested (list …) procedures? If yes, show how and define variable m-5b to hold the result. If no, explain why not.

6a. How would you construct this data structure using the quote operator? In other words, what is the printed representation of this data structure the way the interpreter would print it out? Define a variable `m` with the given data structure using the quote operator.

6b. In this homework download, I supplied a procedure plot-tree that reads Scheme data structures and builds input files for program dot. Program dot reads the input file and generates postscript plots of the data structure. Follow the instructions on the class software page for installing dot and GSView on your system.

Use `(load "plot-tree.scm")` to load my plot program into Scheme, then run `(plot-tree m "hw3-plot.dot")` to generate the dot input file. (This assumes that you defined variable "m" to hold the data structure in part (a) of this question.) Then run dot to generate the postscript file. I have supplied a Windows batch file rundot.bat to do this, and you can read that file in order to see the simple command to type in if you are using a different system. Look at the resulting postscript file with GSView and verify that your commands created the same data structure as shown above.

7. Write a predicate `(present? x m)` that takes an item `x` and a list `m` (possibly having other lists as elements) and checks to see if `x` is present anywhere in `m`, returning `#t` if `x` is found, `#f` if it is not found. `x` may be a symbol or a number. Your procedure should use the numeric = comparison to compare numbers, and eqv? for everything else.

8. A binary mobile consists of two branches, a left branch and a right branch. Each branch is a rod of a certain length, from which hangs either a weight or another binary mobile. We can represent a binary mobile using compound data by constructing it from two branches (for example, using list):

```
(define (make-mobile left right)
  (list left right))
```

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

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

Write the corresponding selectors (`left-branch m`) and (`right-branch m`), which return the branches of the mobile, and `(branch-length b)` and `(branch-structure b)`, which return the components of a branch.

Using your selectors, write a procedure `(total-weight m)` that returns the total weight of a mobile `m`. It may be helpful to define a helper function `branch-weight` that interacts with `total-weight` to calculate the weight of a branch.

9. A mobile is said to be *balanced* if the torque applied by its top-left branch is equal to that applied by its top-right branch (that is, if the length of the left rod multiplied by the weight hanging from that rod is equal to the corresponding product for the right side) *and* if each of the sub-mobiles hanging off its branches is balanced. Write a predicate `(balanced? m)` that tests whether a binary mobile `m` is balanced. It may be helpful to write a helper function `branch-balanced?` that interacts with `balanced?` to determine the balance of a branch.

10. Write a procedure `(apply-all x f0 f1 f2 …` f_n`)` that takes one numeric argument `x` and a variable number of single-argument functions f_i and returns a list that contains one numeric entry for each function, where each entry is the result of applying f_i at `x`.