Homework: Self-referential data

COMP 50

Fall 2013

This homework is due at 11:59PM on **Monday, October 7**.
Submit your solutions in a single file using the COMP 50 Handin button on DrRacket; the homework is the `lists` homework.

All the exercises should be done using the Beginning Student Language *with list abbreviations*.

Data descriptions for this homework

A *letter* is a one-character string containing a single lowercase letter.

A *tile* is either:

- A letter
- The structure `(make-blank)`

A *station* is a structure:

```
(make-station name distance)
```

where `name` is a string and `distance` is a number representing the distance in miles from Boston. You can get real-world information about stations on the Wikipedia page for the Northeast Corridor.

A *1D-tree of stations* is either:

- A station
- A structure (representing a point on the line)

  ```
  (make-point distance south north)
  ```

  where

  - `distance` is the distance of the point from Boston
  - `south` is a 1D-tree of stations, all of which are further away from Boston than the given `distance`
  - `north` is a 1D-tree of stations, all of which are closer to Boston than the given `distance`

In this homework, all lists are made with `empty` and `cons` as described in the book. Selector functions for the `cons` case are `first` and `rest`.

Finger Exercises

For this homework I am recommending the following finger exercises:

- In the first edition, Exercises 9.5.2, 9.5.6, 10.1.2, 10.2.2 (but using strings instead of symbols), 10.2.4, 10.2.5, 11.3.5, 12.2.1, and 12.2.2.
- Define a function that computes how many stations there are in a 1D-tree of stations.
- Define a function that computes the depth of a 1D-tree of stations.

Problems to submit

1. *Define a function* that takes as arguments a distance in miles and a 1D-tree of stations, and which returns information saying the name of the nearest station, how far away it is, and in what direction. You must *write a data definition* that says how the information you return will be represented.

   Use the design recipe for self-referential data (natural recursion). In addition, plan on writing some helper functions that operate on distances and stations. Use your wish list!

   (With any luck, later in the term we will use a similar tree to help you locate US cities quickly.)

   Hint: For purposes of testing, you will do well to define a function that takes a tree and two numbers `lo` and `hi`, and checks the given tree to be sure that

   - All stations have distances between `lo` and `hi`
   - For every `point` in the tree, the stations south and north of that point are located at distances that are consistent with the distance of the point

2. Two fisherman go out in a boat and use nets to pull a whole bunch of bluegills out of the Mystic Lakes. They agree to divide the catch evenly. You are given two buckets and two scales and are charged with placing the fish into the buckets, dividing evenly *by weight*. A friend of Mr Turing's whispers to you that dividing as evenly as possible might take more time than you have. The fishermen agree that your time is worth something, and they will be content if you place one fish at a time in the fairest way possible.

   To prove to the fishermen that your division of fish is reasonably fair, you will use "computational buckets". A computational bucket not only holds a list of fish but also contains a readout that instantly gives the total weight of the fish in the bucket.

   Write a data definition for a pair of computational buckets. The only important thing about a fish is its weight.

   Define a function that takes as arguments a list of fish and returns a pair of computational buckets such that each bucket contains about the same weight of fish as the other bucket.

   Use the design recipe for self-referential data (natural recursion).

   It turns out that the fairness of your division can be strongly affected by the order in which you consider the fish. Please create a random list of 30 fish and *run three experiments*:

   - Divide the fish
   - Sort the list of fish in *decreasing* order of weight, then divide them
   - Sort the list of fish in *increasing* order of weight, then divide them

   How do the experiments affect the fairness of the division?

   A convenient way to report on the experiments is to produce a list of three strings, as in the following example:

```
> (three-experiments (N-random-fish 30))
`("When divided, randomly ordered fish give buckets that differ by [REDACTED]"
  "When divided, increasing fish give buckets that differ by [REDACTED]"
  "When divided, decreasing fish give buckets that differ by [REDACTED]")
```

Hint: to sort the fish, you will want to use the insertion sort defined in section 12.2 of the first edition textbook. You can develop a similar sort function that sorts in increasing order.

Karma problem

A. *Define a function* that consumes a list of stations and produces a 1D-tree containing those same stations. The 1D-tree must be balanced, which in this case means that its depth must be at most one more than the base-2 logarithm of the number of stations.