CS5001 Assignment 8

Programming Language: ISL with Lambda

Due Dates: Thursday 3/22 @ 10:00pm

This entire assignment is extra credit. Not doing it will have no negative impact on your grade. Extra credit will be assigned per-exercise, so completing only one of the exercises is okay.

Problem 1

Design `cartesian-product`, which takes a list of lists and returns a list of lists.
The following test should pass:

```
(check-expect (cartesian-product '((0 1 2) (3 4)))
              '((0 3) (0 4) (1 3) (1 4) (2 3) (2 4)))
```

(You may want to look up the single-quote list abbreviation used in the `check-expect` above.) Given the lists `'(0 1 2)` and `'(3 4)`, the function outputs a list of lists, each element of which contains two elements, where the first element came from the first list and the second came from the second list. The output is also in an order such that everything beginning with `0` comes first, and everything ending with `3` comes before all that ends with `4`. Now, for a more formal specification...

The Cartesian product of lists `L-1, L-2... L-n` is defined as a list of lists where each inner list is of size `n`, where the `k`th element of `a`, a list in the output sequence, is from `L-k`. It is ordered in such a way that, if `L-1` is the list `x-1, x-2, x-3...x-y`, all output lists beginning with `x-1` appear before `x-2`, appear before `x-3`, etc. Then, within all lists that begin with `x-1`, the ordering is defined the same way, except with `L-2` taking the part of `L-1`, and so on.

Here's another example to illustrate:

```
(check-expect
  (cartesian-product '(a b) (red green blue) (circle square)))
 '((a red circle)
   (a red square)
   (a green circle)
   (a green square)
   (a blue circle)
   (a blue square)
   (b red circle)
   (b red square)
   (b green circle)
   (b green square)
   (b blue circle)
   (b blue square)))
```

Problem 2

```
;; A LeafyBinaryTree (LBT) is one of:
;; - 'leaf
;; - (make-node LBT LBT)
(define-struct node (left right))
```

Design `all-leafy-trees` which consumes a natural number `n` and creates a list of all leafy binary trees of height `n`.

Here are a few tests that should pass:

```
(check-expect (all-leafy-trees 0) (list 'leaf))

(check-expect (all-leafy-trees 1)
              (list (make-node 'leaf 'leaf)))

(check-expect (all-leafy-trees 2)
      (list
            (make-node (make-node 'leaf 'leaf) 'leaf)
            (make-node 'leaf (make-node 'leaf 'leaf))
            (make-node (make-node 'leaf 'leaf) (make-node 'leaf 'leaf))))
```