Binary Mobiles

Exercise 2.29 in *Structure and Interpretation of Computer Programs* (the "Wizard Book") starts, "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." Later, it gets to the crux of the problem, "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 submobiles hanging off its branches is balanced. Design a predicate that tests whether a binary mobile is balanced."

In Haskell, we can define the mobile as a recursive data structure:

```
type Rod = Integer
type Wt  = Integer

data Branch =
     Simple Rod Wt |
     Complex Rod Mobile
     deriving Show

data Mobile = Mobile Branch Branch
              deriving Show
```

The words "deriving Show" tell Haskell to construct a string representation for a Mobile.

Now we can construct a Mobile and write its string representation to a file, but how can we use the representation to reconstruct the Mobile? One way is to write a grammar for the representation for the representation of a Mobile. Then we can write a parser for the grammar, but we also have to construct to the Mobile represented.

It's straightforward to do that in Haskell, using a technique called "parser combinators." (See http://eprints.nottingham.ac.uk/223/1/pearl.pdf.) A combinator is a function which takes a function as a parameter and produces a function as output. (The technical definition is a bit different, but this will do.) The Haskell definition of a parser is

```
type Parser a = String → [(a, String)]
```

or, as Dr. Seuss would put it,

A parser for things
Is a function from strings
To lists of pairs
Of things and strings[1]

Parser combinators make it easy to combine parsers. In the program I have supplied, you can see that lexical analysis, parsing, and constructing the Mobile are all combined into a single process, by gluing together short, simple functions. For example, once we have a parser named "mobile" that parses a single mobile, we can construct a parser for a comma-separated list of Mobiles enclosed in square brackets as:

```
mobiles = brackets (commaSep mobile)
```

You have only to implement the function isBalanced, which tells whether or not a binary mobile is balanced.

[1] Fritz Ruehr, quoted in *Programming in Haskell*, by Graham Hutton