Advanced Programming (CSE 399)
Homework Assignment 4

Due Tuesday, February 8, at noon

A collection of files that form the starting point of this assignment is available on the Schedule page under the main course web page. Begin by grabbing these files, unpacking them, and making sure that you can successfully run the main program (in `Main.hs`). Remember that you need to use Hugs for this assignment.

1. Before starting this problem, read Sections 24.1 to 24.4 of SOE.

   Here is a slightly more refined variant of the `SimpleXML` datatype used for the last assignment.

```
data XML =
         PCDATA  String
       | Element ElementName [Attribute] [XML]

type ElementName = String
type Attribute = (String,String)
```

   Use the facilities described in 24.4 to create an `instance` declaration making the `XML` type an instance of the `Show` class. Use standard XML concrete syntax for printing `SimpleXML` structures. For example, executing

```
xml = Element "A" []
          [PCDATA "data1",
           Element "B" [("a1","v1"),("a2","v2")] [PCDATA "moredata"],
           PCDATA "data2",
           PCDATA "data3",
           Element "C" [("a3","v3")] [],
           PCDATA "data4"]

main = print xml
```

   should result in the following output on `stdout`:

```
<A>data1<B a1="v1" a2="v2">moredata</B>data2data3<C a3="v3"></C>data4</A>
```

2. Before starting this problem, make sure you thoroughly understand SOE chapter 13 (you can just skim the kaleidoscope example). You may want to play around with variants of some of the examples in the chapter before starting on the following steps.

   The file `Main.hs` includes a definition of a very simple planetary system containing just a single object called `sun`.

   (a) Use the provided function `translateB` to write a function

   ```
   orbit :: Behavior Picture     -- the satellite
         -> Behavior Picture     -- the fixed body
         -> Float                -- the frequency of the orbit
         -> Float                -- the x-radius of the orbit
         -> Float                -- the y-radius of the orbit
         -> Behavior Picture
   ```

   that takes two picture behaviors and makes the first orbit around the second at the specified distance and with the specified radii. That is, the two pictures will be overlayed (using `over`) and, at each time t, the position of the satellite will be translated by $\texttt{xradius} \times cos(t \times \texttt{frequency})$ in the x dimension and by $\texttt{yradius} \times sin(t \times \texttt{frequency})$ in the y dimension.

   Test your function by creating another circle, `mercury`, colored red and with radius 0.1, and making it orbit around the `sun` with a frequency of 2.0, and with radii of 2.0 and 0.2 in the x and y axes, respectively.

   (b) Use your `orbit` function again to create an earth with a moon orbiting around it. The earth should be a blue circle of radius 0.2. The moon should be a white circle of radius 0.08. The orbit of the moon around the earth should be characterized by a frequency of 2.5 and radii of 0.5 and 0.15. Use `orbit` once again to make this whole system orbit around the sun (in addition to `mercury`).

   (c) A problem you might have noticed is the overlay behavior of planets. For this part modify orbit to put planets over or under each other. Hint: you might find the lifted conditional *cond* from SOE useful for this part.

   (d) Modify your functions (and write any support functions that you find necessary) to make the orbital distances and planet sizes shrink and grow by some factor (you can pass this factor as parameter to the `orbit` function), according to how far the planets are from the observer. For example, the earth and moon should look a little smaller when they are going behind the sun, and the orbital distance of the moon from the earth should be less.

   Choose the scaling factor so that the solar system simulation looks good to you.

   (e) **Optional:** Add some other planets, perhaps with their own moons. If you like, feel free to adjust the parameters we gave above to suit your own aesthetic or astronomical tastes. Make sure, though, that the features requested in parts (c) and (d) — growing, shrinking, occlusion, etc. — remain clearly visible.

3. **Extra credit:** Figure 13.3 in SOE defines an equality function for behaviors that fails whenever it is called. This sort of thing represents both an unfortunate weakening of the type system (the compiler will allow us to write a program that accidentally compares behaviors, even though this makes no sense) and a danger signal for the design: it generally means that something is not structured properly. Why do we need it here? Are there similar problems hiding in any of the other numeric classes? Could the numeric classes be restructured to fix this problem? Would it be a good idea to do so?

Submission instructions:

- Put the solutions to all parts into the file `Main.hs`. (If you choose to do the extra credit part, either write your solution as a comment in `Main.hs` or, if you prefer to use a word processor, email a separate PDF document.)

  Your `main` action should look exactly like the one provided in our `Main.hs`:

  ```
  main =
    do print xml
       animateB "Solar system" planets
  ```

  Email (just) the file `Main.hs` to `bcpierce@cis.upenn.edu`.

- As usual, make sure this file is accepted by Hugs without errors and follows the other rules in the Style Guide on the course web page. In particular, please put your name in a comment at the top of the file. And remember that style counts!