PLC: Homework 2 [100 points]

Due date: Wednesday, February 20th
3 extra-credit points if you turn it in by Tuesday, February 19th

About This Homework

For this homework, you will gain further practice with higher-order functions, pattern-matching, and list operations. You will also gain further practice implementing languages, in this case a language called ShapeLang for describing graphics. You will also begin using the Functor, Applicative, and Monad type classes.

I am posting the homework with still some more problems to write, so you can get started. I will update the homework soon with the remaining problems.

How to Turn In Your Solution

You should create a directory called `hw2` (exactly this!) in your personal repository, and add your Haskell files to this directory. Then push the directory (and all your Haskell files) up to github.uiowa.edu.

As for hw0, you can check that you have submitted correctly by going to the URL for your subversion repository. Also, as for hw0, please use exactly the file names we are requesting (so do not change the names of these files).

Partners Allowed

You may work by yourself or with one partner (no more). See instructions for hw1 on the protocol you should use if you do work with a partner.

How To Get Help

You can post questions in the `hw2` section on Piazza.

You are also welcome to come to our office hours. See the course's Google Calendar, linked from the github.uiowa.edu page for the class, for the locations and times for office hours.

1 Reading

Read Chapters 7, 8 and 12 of *Programming in Haskell*.

2 Positional Notation [35 points]

Positional notation ("base-n notation") is the general scheme for representing numbers that encompasses the decimal notation we standardly use. In this problem you will implement some operations related to positional notation. A number in positional notation is a list of digits. We will write these from least to most significant. So we will write 134 as the list `[4,3,1]` in our Haskell code. The code for this section is in `Positional.hs`. The problems below are each worth 7 points each. See Chapter 7 for some inspiration.

1. Implement the `toPos` function which takes an integer x and a base b, and converts x to base-b positional notation. So if x is 8 and n is 2, `toPos` should return `[0,0,0,1]` as this is the base-2 (i.e., binary) representation of 8.

2. Implement `fromPos` that takes a number in positional notation and a base, and returns the integer which that positional notation represents. So `fromPos b` undoes what `toPos b` did (it will return the starting number x).

3. Next we will convert `Pos` to `Expr`, a type for basic arithmetic expressions. For this, first make `Expr` an instance of the `Show` typeclass. Your `show` function for `Expr`s should always put parentheses around plus and times expressions (only), **not** trying to omit them in some cases.

4. Fill in the definition of `toExpr` to convert a `Pos` to an `Expr`, where you show the sums of powers of the base explicitly. For example, `toPos 3 11` should return `[2,0,1]`, which your `toExpr` function should convert to an `Expr` that prints as:

   ```
   ((2 * 1) + ((0 * 3) + (1 * 9)))
   ```

5. Define `addPos` so that it takes in a base and two positional representations, and returns the sum of those representations as another legal positional number with respect to that base. Your algorithm is not allowed to convert the positional numbers to Integers, add, and convert back. Instead, you should implement the grade-school algorithm for adding positionally represented numbers with a carry.

3 Stars in Scalar Vector Graphics [39 points]

In this problem, you will implement code to generate a description of a star in Scalar Vector Graphics (SVG) format. Example stars generated by my solution are provided in the `sample-output` directory. Of course you are not expected to know the SVG format for this problem. You can glean everything you need to know from the sample output or online. If you open one of the sample output files in a web browser, you will see the rendered star. The provided `Tests.hs` generates those stars from your solution (it just generates files with a blue circle by default until you write the code below).

Your goal in this problem is to implement the `showStar` function, which takes in the following inputs:

- tx: amount along the x-axis to translate the star, where the top left corner of a page is coordinate (0,0).
- ty: amount in the y-axis to translate the star.
- r: the radius of the star (distance from the center to each point of the star)
- sep: the "separation" (my term, not sure if there is a standard one), which controls how thick the points of the star are
- n: the number of points of the star.

Of course you will have to write a number of helper functions, that seek to generate the coordinates of the lines connecting the points of the star. Here is a high-level description of how to do this:

1. Generate n evenly spaced points around a circle of radius r. To generate the k'th point, you use the coordinates
$$(r * (cos((k * 2\pi)/n)), r * (sin((k * 2\pi)/n)))$$
Amazingly, all those functions are available in the Haskell prelude including `pi`! You will need to convert integers to floats using `fromIntegral`, and from floats to integers using `round`. Be careful about where you put your calls to `fromIntegral` so you do not truncate your floating point computation too early (resulting in the wrong points).
2. Generate a list of edges, where an edge is a pair of points. You should do this by connecting the k'th point from the ones you generated around the circle, to the $k+sep$'th point (wrapping around to the front of the list of points if you exceed n).
3. Show each edge as an SVG `line` (you can see the `sample-output` files for examples).
4. `showStar` can then put the above functions together to generate a string with all the `line`s. The provided `writeStar` function will then call this with appropriate HTML boilerplate to show the SVG.

A correct solution is worth 39 points. If you have trouble finishing, you can just generate the points around the circle as a polygon, as in `sample-output/polygon.html`. This easier alternative is worth 20 points. (You would change your `showStar` to show these polygons, so your output would still go in `star1.html` etc. when the `main` function of `Tests.hs` is run. You would just have polygons in those files instead of stars.)

Bonus: create a file called `FancyStar.hs` with a `main :: IO ()` function that calls a function `showFancyStar` to create `fancyStar1.html` through `fancyStar4.html`. Fancy stars are whatever you want them to be, but have to have some extra wrinkles on top of plain old stars. We will look at submitted fancy stars in class (non-anonymously) [4 bonus points].

4 Functors and the IO Monad [25 points]

In `FuncEx.hs` fill in the following [5 points each]:

1. implement `map2` to map a function down two levels of list structure; similarly `map3`.

2. implement `mapTree2` to do the same sort of thing as `map2` but for the `Tree` data structure.
3. implement `print2` that takes in two showable things and prints them out on two separate lines, using the IO monad.
4. Define a function `fmap2` with its type, which can fmap any function through two layers of a `Functor` f. Specialized to lists, this will be equivalent to the `map2` function in `Basics.hs`, and also to `mapTree2`.
5. Fill in the definition of function `printShowables` so that given a list of showable values, it prints each of them using `putStrLn`, in the `IO` monad.