Advanced Programming (CIS 552)
Homework Assignment 1

Due Wednesday, January 23, at 2PM

Preliminaries, part 1

1. Read chapters 1 to 3 of the Haskell School of Expression. (Pages 30-33 can be skimmed.)
2. Get yourself a running Haskell installation, either on ENIAC or on your own machine.
   (a) If you want to use the GHC compiler installed on ENIAC, add the line

   ```
   export PATH=$PATH:/home1/b/bcpierce/ghc-6.8.2/bin
   ```

   to your `.bashrc` file. (If you are using a shell other than `bash`, you'll probably need a slight variant of this.) Type `bash` to create a new shell and load the updated `.bashrc`. Type `which ghci` to make sure that things have been set up correctly.
   (b) If you prefer to install Haskell on your own machine, visit the Glasgow Haskell Compiler home page (Google `GHC`) and download and install the appropriate set of binaries for your architecture. Make sure you get version `6.8.2`.
3. Test your (or our) Haskell installation:
   (a) Run the Haskell interpreter by typing `ghci` and verify that evaluating `2+2` yields `4`.
   (b) Create a file `test.hs` containing a single line:

   ```
   f x = x + x
   ```

   (c) Load this file by typing

   ```
   :load "test.hs"
   ```

   to `ghci`. (Alternatively, quit `ghci` and restart it by typing `ghci test.hs`.)
   (d) Evaluate `f 5` to test that this worked.

Assignment, part 1:

1. Create a file `<yourlastname>1.hs` containing the definitions of the `Shape` datatype and the functions over shapes from Chapter 2 of SOE. Test these definitions with `ghci`.
2. In the same file, define functions `rectangle` and `rtTriangle`, as suggested at the end of Section 2.1 (Exercise 2.1) of SOE. Each should return a `Shape` built with the `Polygon` constructor.
3. Write a function `sides :: Shape -> Integer` that returns the number of sides in a given shape. For purposes of this problem, you may assume that every ellipse has 42 sides.
4. Write a function `bigger :: Shape -> Float -> Shape` that takes a shape `s` and an expansion factor `e` and yields a new shape that is the same as `s` except bigger by a factor of `e`.

5. The *towers of Hanoi* is a simple recursive game. (You've probably seen it before, so I won't provide a lot of detail here; if it is not familiar, you may want to read a little about it on the web to get some intuition.) The purpose is to move a stack of n discs from one peg to another, using a third peg as "temporary storage" and respecting the constraint that no disc is ever placed on top of a smaller one. It is played as follows:

   To move n discs (stacked in increasing size) from peg a to peg b using peg c as temporary storage...

   (a) move n discs from a to c using b as temporary storage

   (b) move the top disc from a to b

   (c) move n discs from c to b using a as temporary storage.

   Write a function `hanoi :: Int -> String -> String -> String -> IO()` that plays towers of Hanoi. For example, when invoked like this

   ```
   hanoi 2 "a" "b" "c"
   ```

   it should yield an action that, when executed, prints this:

   ```
   move disc from a to c
   move disc from a to b
   move disc from c to b
   ```

Preliminaries, part 2:

1. Download the `SOE` code bundle from the School of Expression home page (Google `Haskell SOE`) and unpack it somewhere convenient.

   If you've installed GHC on your own machine, you will first need to install the GLFW library and—depending on your OS—possibly some lower-level support packages. See the instructions on the SOE download page.

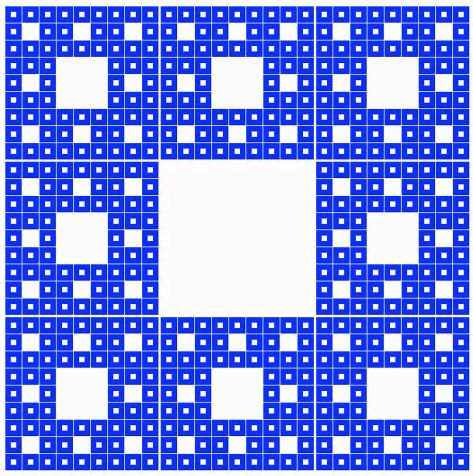
2. Test that things are working. Going to the `SOE/src` directory and typing

   ```
   ghci Draw.lhs
   *Draw> main0
   ```

   should display a graphics window with a couple of shapes in it.

Assignment, part 2:

1. The *Sierpinski Carpet* is a recursive figure with a structure similar to the Sierpinski Triangle discussed in Chapter 3:

   Write a function that displays this figure on the screen.

   Note that you'll need to either run your program in the directory `SOE/src` or else set up GHC's search path to include this directory.

   Also, the organization of the SOE graphics libraries has changed a little since the book was written: your program should begin with `import SOE`, not `import SOEGraphics`.

2. Use the `SOE` graphics library to create your own fractal image.

   This part is intentionally open-ended! The only requirement is that your image should have some sort of recursive self-similarity. Feel free to do some reading about fractal images if you like (Wikipedia has many relevant articles) or just experiment until you find something you like.

Submission instructions:

- Collect your solutions into a *single file* of Haskell source code.

  The file should define an action `main` that, when executed, displays your individually designed fractal image.

- Make sure this file is accepted by `ghci` without errors.
- Email this file to `jschorr@seas.upenn.edu` before the deadline.
- Email a screen shot of your fractal image to `bcpierce@cis.upenn.edu` before the deadline.