PLC: Homework 2 [100 points]

Due date: Wednesday, February 21st, 9pm
3 extra-credit points if you turn it in by Tuesday, February 20th, 9pm

About This Homework

For this homework, you will practice 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 over the weekend (Feb 10-11) 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 add the directory (and all your Haskell files) to subversion. You should copy the files from the `hw2` directory of the course repo to your own `hw2` directory, before you start modifying them. We may release changes to the original homework files if a bug is reported, for example, so you do not want to modify the original files, just your copies in the `hw2` directory of your personal repo.

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.

Extra credit for Piazza help

Similarly to previous homeworks, we will give 2 extra-credit points if you provide a correct and useful answer to another student's question on Piazza. If you do this, just enter the Piazza link number for the question (it looks like `@5`; you can find it if you hover over the little downward triangle by the post title in the panel on the left side of the page) in a file called `piazza-help.txt`, in your `hw2` directory.

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 Resources tab of the Resources page on Piazza, for the locations and times for office hours.

1 Reading

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

2 Basic Problems [60 points]

You will fill in functions in `Basics.hs` and `SnocLists.hs`. Similarly to `hw1`, if you load the file `Main.hs` either in `haskell-mode` in `emacs` or by loading it into `ghci`, you can run a suite of tests I am including for your code for this problem, and check that the answers seem correct. You will see `Prelude.undefined` for functions you have not defined yet, when you run `Main`.

1. Add an instance declaration making `Tree` (from the `Data.Tree` module, from Haskell's libraries) an instance of `HeadClass.head` (which we were working with in class on Jan. 30th; see the lecture materials from that day for examples). Uncomment a line in `Main.hs` as it says there when you have done this. [5 points]
2. Add an instance declarations making `SList` a `Functor`. I am providing a function called `smap` in module `SnocLists` that you can use for this. [5 points]
3. implement `map2` to map a function down two levels of list structure; similarly `map3` [5 points each].
4. implement `mapTree2` to do the same sort of thing as `map2` but for the `Tree` data structure [10 points]
5. implement `print2` that takes in two Showable things and prints them out using the IO monad [10 points]
6. Fill in the definition of `evalExplicit` that will evaluate an `Expr` (as defined in `Expr.hs`). `evalExplicit` takes in an integer to use as the divisor when evaluating a `Div` expression. So `evalExplicit (Div (Num 7)) 3` should evaluate to 2. [10 points]
7. implement `toSlist` in `SnocLists`, to convert a list to a snoc-list. [10 points]

3 Intermediate Problems [30 points]

Fill in functions in `Intermediate.hs` and `SnocLists.hs`:

1. 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`.Uncomment a line in `Main.hs` as it says there, when you have done this.[10 points]
2. Fill in the definition of `showSlist` so that `SLists` are printed with the same syntax as lists, but with the head at the right of the string. I found I needed a helper function for this. [10 points]

3. Fill in the definition of function `printShowables` in `Intermediate.hs` so that given a list of showable values, it prints each of them using `putStrLn`, in the `IO` monad. [10 points]

4 Challenge Problems [10 points]

You will fill in code in `Challenge.hs`:

1. Fill in the definition of `postorder` so that it returns a snoc-list representing the post-order traversal of the given input `Tree` (look online if you forgot what a post-order traversal is). [5 points]
2. Using the `Applicative` type `Q` defined in `Challenge.hs`, fill in the definition of `evalImplicit` (you will likely need a helper function) so that it implicitly passes the divisor to use for evaluating `Div` expressions. So the code will give the same resulsts as `evalExplicit`, but using the `Applicative` type to avoid explicitly passing the divisor. [5 points]