PLC: Homework 1 [100 points]

Due date: Wednesday, February 7th, 9pm
3 extra-credit points if you turn it in by Tuesday, February 6th, 9pm

About This Homework

For this homework, you will practice writing functions by recursion and pattern-matching in Haskell.

How to Turn In Your Solution

You should create a directory called `hw1` (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 `hw1` directory of the course repo to your own `hw1` 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 `hw1` 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). Only one partner should submit the solution by adding the Haskell files to his/her personal repository. This submitting person should also commit (to the `hw1` directory) a file called `ack.txt` (for "acknowledgment"), which contains just the Hawkid of the other (non-submitting) partner (no other text). The non-submitting partner should then add just one file to his/her repository, called `partner.txt`, which should contain just the Hawkid of the submitting partner. This is to ensure that both partners agree that they are submitting the solution together. You are free to divide up the problems and tackle them separately, or to work on problems together.

How To Get Help

You can post questions in the `hw1` 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 3, 4, 5, and 6 of the required book, *Programming in Haskell*, by Graham Hutton.

2 Basic Problems [67 points]

You will fill in functions in `Colors.hs` and `BoolK.hs`. 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. Fill in definitions for the `undefined` functions in `Colors.hs`, based on subtractive color theory. They are worth 7 points each.
   - complementary colors: yellow and purple, red and green, blue and orange.
   - yellow and red are the constituents of orange; red and blue of purple; and yellow and blue of green.
2. Fill in definitions for the `undefined` functions in `BoolK.hs`, for Kleene's 3-valued logic. These are worth 8 points each. You can define the functions by pattern matching, or in terms of functions that are already available. Either way is fine.
   - `equivK` is for implication (if).
   - `equivK` is for equivalence (iff). Be careful, though: `UnknownK` is not equivalent to `UnknownK`, because each `UnknownK` stands for a possibly different unknown value. So the first `UnknownK` could be in reality True, and the second could be False.
   - For `showBoolK`, please print TrueK as "true", FalseK as "false", and UnknownK as "unknown".

3 Intermediate Problems [21 points]

In `SnocLists.hs`, you will find a definition of a recursive datatype `SList`, representing lists where the head is the second argument and the tail is the first to the `Scons` constructor (corresponding to the usual `(:)` constructor for Haskell's built-in lists).

1. Fill in the definitions of `sappend`, `slength`, and `smap`, corresponding to `(++)` (append), `length`, and `map` on Haskell's built-in lists. Each function is worth 7 points.

4 Challenge Problems [12 points]

1. Fill in the definitions of `sfilter`, `sintersperse`, and `sconcat`, so that they behave in corresponding manner to the Haskell list operations `filter`, `intersperse`, and `concat` (see the

documentation for these functions on Hoogle). These are worth 2 points each.

2. In a file called `More.hs` (which you must create), define a function called `ascendings` which, given a list of elements of type `a` where `a` is in the `Ord` type class, return a list of lists of `a` elements, consisting of the maximal ascending subsequences of the input list. For example, given `[1,2,3,2,4,1,100]`, your function should return `[[1,2,3],[2,4],[1,100]]`. Getting the correct type for this (which you should write as part of your code) is worth 2 points, and the correct code is then 4 more points.