Problem Set #5

due October 15

Problem 1. The problem practices interactive programming and IO. You will implement the word game Connection `https://www.nytimes.com/games/connections` via a simple text-based interface similar to the games we studied in class. For those of you unfamiliar with the game, here is the Wikipedia reference as well: `https://en.wikipedia.org/wiki/The_New_York_Times_Connections`.

You can design your interface and text-based graphics as you wish, however, for the purposes of running the game on Submitty your game should follow these interactions:

- Player 1 enters the name of a file, e.g., `puzzle477.txt`, that stores the puzzle in the following format:

  ```
  Y:COOK_WITH_HEAT_AND_WATER BLANCH BOIL POACH STEAM
  G:COMMON_PERFUME_INGREDIENTS AMBERGRIS MUSK ROSE VANILLA
  B:CHARACTERS_WITH_PET_DOGS CHARLIE DOROTHY SHAGGY WALLACE
  P:CAPITAL_CITY_HOMOPHONES KETO ROAM SOPHIA SOUL
  ```

- Next, the system initiates play with Player 2. Each play round is as follows:
  (1) System displays the state of the game: categories Player 2 has guessed correctly, shuffled board of remaining words, remaining mistakes.
  (2) System prompts Player 2 to enter a guess of four words, each word on a new line.
  (3) System evaluates new guess. Suppose guess is incorrect; if there are no remaining mistakes Player 1 wins, otherwise system updates number of mistakes and goes back to (1). Suppose guess is correct; if there are no remaining words Player 2 wins, otherwise system updates state of the game and goes back to (1).

Note: Some minimal starter code is provided in `Ps5Connections.hs`. Do not use additional imports.

Problem 2. In this problem you will develop Vector, an array-like data structure that stores a sequence of elements. One can index into a vector as well as insert and delete elements. Other operations on vectors are the append operation as well as `fmap` and `>>=` as vectors are instances of `Functor` and `Monad`. You will build on the previous homework which emphasized `Foldable` and `Monoid` operations, but will code `fmap` and the monadic bind as well.

```
class (Monad v, Foldable v, forall a. Monoid (v a)) => Vector v where
    first :: v a -> Maybe a
    final :: v a -> Maybe a
    index :: Int -> v a -> Maybe a
    insert :: Int -> a -> v a -> Maybe (v a)
    delete :: Int -> v a -> Maybe (v a)
```

The starter file `Ps5Vector.hs` gives additional instructions for where to add code. The homework is roughly divided in three parts:

(1) Define basic list-like operations in terms of the Monad, Foldable, and Monoid operations.

(2) Instantiate Vectors as lists. Implementing Vectors as lists gives us a constant `first` operation, however all other operations are $O(N)$ in the worst-case where N is the size of the vector.

(3) Instantiate Vectors as balanced trees. In this case the underlying structure is the AVL tree, a balanced binary search tree. As you know from Algorithms class, balanced trees trade $O(1)$ `first` operation for $O(lgN)$ for all operations. Ideally, you will implement the re-balancing functionality for the AVL tree structure, however, if you don't have time you can omit it and focus on indexing, inserting and deleting elements. The balance property is worth only a few points.

Note: Use the `Control.Monad` and `Control.Applicative` imports, but try not to include additional imports. When done submit `Ps5Vector.hs` in Submitty.

Haskell Style Guide. Adapted from Stephanie Weirich (UPenn CIS 5520).

- **Write a type signature for every function.** We will be strict about this when grading. Hint: try writing the signature *before* writing the function. If you do write the function first, try deducing the signature and if this doesn't work, there is always the `:t` command.
- Make sure that your code produces no errors or warnings. Code with errors receives 0 on Submitty and we will mark down warnings during TA grading.

- Use consistent indentation.
- Do not use tab characters, use space for indentation. GHC should be flagging tabs, but nevertheless, be careful.
- No line should have more than **80 characters**.
- Use whitespace to make your code readable. Add whitespace on either side of binary operators, e.g., write `3 * n + 1` instead of `3*n+1`.

- Use descriptive names.
- Follow standard Haskell naming conversions: (1) use camelCase for compound names, and (2) use `x` and `xs` when you pattern-match lists.

- Use comments. Each function definition should be preceded by a comment.
- Comment should say what the function does, not how.
- Comments should be concise. Do not overcomment.
- Use full English sentences.

- Do not leave incomplete pattern matches. They will be marked down.
- Tuples, records and datatypes can be decomposed. You can also use the `@` operator if you need a reference to both the object and its components.
  For example, do not use this:

```
f arg1 arg2 = ...  where
   x = fst arg1
   y = snd arg1
   z = fst arg2
```

  Use this instead:

```
f (x,y) (z,_) = ...
```

- Combine nested case expressions.
  For example, do not use this:
  ```
  case x of
        Red -> case y of
                 Red  -> True
                 Blue -> False
        Blue -> case y of
                 Red  -> False
                 Blue -> True
  ```
  Use this instead:
  ```
  case (x,y) of
        (Red,   Red) -> True
        (Blue, Blue) -> True
        (   _,    _) -> False
  ```
- Use library functions, unless the assignment explicitly forbids them. Use Haskell's search engine Hoogle to look up library functions.