Exercise 9: Type Checking a "Spreadsheet"

This exercise provides a light introduction to the world of type checking and type inference. We will be type checking a data structure similar to a "spreadsheet" or a SQL table.

The way that we perform type checking in this exercise is quite different from the *dynamic* contract checking that we did in Assignment 1. In particular, we will be type checking spreadsheet formulas without knowing anything about the *values* in our spreadsheets. In other words, we will not be evaluating any code to perform our type checking.

Deadline: November 19, 2019 before 10:00pm

Starter code

- `Ex9.hs`
- `Ex9Types.hs`

You will only be submitting `Ex9.hs`. Do not make any modifications to `Ex9Types.hs`. We'll be supplying our own version to test your code.

Spreadsheet Data Structure

Consider the *metadata* of a spreadsheet data structure or a SQL table. This metadata describes the kinds of data stored in each column of our spreadsheet. Each column has a:

- name (String): describing the content of this column
- type (Type): the type of the column, either a NumCol or a StrCol
- formula (Maybe Formula): either `Nothing` if the column contains raw data, and a `Formula` that describes how to compute this column based on other columns.

Here is an example of a spreadsheet metadata:

first_name	last_name	full_name
type: StrCol formula: Nothing	type: StrCol formula: Nothing	type: StrCol formula: `Just (Concat (Column "first_name") (Column "last_name"))`

In Haskell, we will represent a spreadsheet metadata as two separate `Map` objects:

```
type TypeEnv = Map.Map String Type
type Formulas = Map.Map String (Maybe Formula)

exampleTypeEnv :: TypeEnv
exampleTypeEnv = Map.fromList [
    ("first_name", StrCol),
    ("last_name", StrCol),
    ("full_name", StrCol)]

exampleFormulas :: Formulas
exampleFormulas = Map.fromList [
```

```
("first_name", Nothing),
("last_name", Nothing),
("full_name", Just (Concat (Column "first_name") (Column "last_name")))]
```

Task 1: Type Checking

The only task in this exercise is to check that the types and formulas specifying a spreadsheet is consistent. This check is possible because all of our formulas are typed. We will be working with the following formulas:

- `Plus`: which takes two numeric values and produces a numeric value
- `Concat`: which takes two string values and produces a string value
- `Length`: which takes a string values and produces a numeric value
- `NumToString`: which takes a numeric values and produces a string value
- `Column`: which duplicates (or references) a column, and the type of the new value is identical to the original

Formulas are recursive, so it is possible to have complex formulas like this:

```
(NumToString (Length (Column "first_name")))
```

Complete this task by writing the `typeCheck` function, which takes a `TypeEnv` map and a `Formulas` map, and returns either True or False.

You may assume that the two arguments to `typeCheck` contain the same set of keys (same column names). You may also assume that the formula for a column does not reference itself, and that there will not be mutually recursive formulas.

You may find the functions in `Data.Map` helpful, and can read about them here: https://hackage.haskell.org/package/containers-0.4.2.0/docs/Data-Map.html The functions `Map.lookup` and `Map.foldlWithKey` are especially relevant.