Assignment 1

CS 4400 Programming Languages

Due: Saturday, September 19, 2020, 9pm

Submission:

1. Submit one file named `Assignment01.hs` via https://handins.ccs.neu.edu/courses/119.
2. At the very top, the file should contain a preamble following this template.

```
{- |
Module      :  Assignment01
Description :  Assignment 1 submission for CS 4400.
Copyright   :  (c) <your name>

Maintainer  :  <your email>
-}

module Assignment1 where

... your code goes here ...
```

   The rest of the file will contain your solutions to the exercises below.

3. Every top-level definition[1] must include a purpose statement (for functions) and a type signature, followed by one or more defining equations.
4. Double-check that you have named everything correctly.
5. Make sure your file loads into GHCi or can be compiled by GHC without any errors.

Purpose: The purpose of this assignment is to get a bit of practice with Haskell, especially with writing functions (with type signatures and purpose statements) and working with some datatypes. Most of the concepts involved in the exercises here should be familiar to you from Fundies 1 or other parts of the curriculum. However, the Haskell specifics might be new. If something is not clear, you are encouraged:

[1] A *top-level definition* is one that is not nested inside another one.

a) to look at online resources – see the course page for some suggestions; and
b) to ask questions after class, during office hours, or on Piazza.

We also recommend familiarizing yourself with Hoogle – a very handy search engine for Haskell's libraries. It allows searching by name or by type.

Readability will initially play a small role, but will become more important with each further assignment. I repeat, however, that purpose statements and signatures for top-level definitions are required.

Tasks

1) If you haven't yet, register on Handins: https://handins.ccs.neu.edu/courses/119
2) If you haven't yet, register on Piazza: https://piazza.com/northeastern/fall2020/cs4400 (class code is `MONAD`)
3) Define a *recursive* function `isOdd` which returns `True` if the given integer is odd and `False` otherwise. Do not use any divisibility tests. You can assume that the argument will be greater than or equal to 0.
4) Write a function `binToInteger`, which takes an arbitrarily long list of booleans representing a binary number and returns its integer representation. The binary is given in the reverse order, that is, the least significant bit is first in the list. E.g., 1 is `[True]`, 1010 ($= 10$ in decimal) is `[False, True, False, True]`.
5) Define a function `isDescending`, which determines whether the given list of integers is sorted in descending order. Do not use any sorting operation.
6) A `BinaryTree` is either

   - `Empty`, or
   - a `Node` with an integer element, a left subtree, and a right subtree – in that order.

   (a) Define a Haskell datatype representing the above tree, with the correct constructors and argument types.

   (b) Define a function `sumTree`, which takes a `BinaryTree` argument and returns the sum of all its elements.