CMSC-16100

Honors Introduction to Programming, I
Autumn Quarter, 2020

Lecture 3: Abstract Data Types

*__Exercise 3.1__ Consider the collatz function defined as follows:

$$\text{collatz}(n) = \begin{cases} 1, & \text{if } n = 1 \\ 1 + \text{collatz}(n/2), & \text{if } n \text{ is even} \\ 1 + \text{collatz}(3n+1), & \text{otherwise} \end{cases}$$

Give Haskell definitions of `collatz` using guards, and `collatz'` using `if ... then ... else ...`. [We can use apostrophes in variable names!] Note that you should use `div` rather than `(/)` to divide integral values.

Exercise 3.2 The `Prelude` provides some simple functions for dealing with pairs, `fst` and `snd` for extracting components, and `curry` and `uncurry` for swizzling between functions that expect two arguments, either separately, or packaged together via a pair.

Exercise 3.3 We could have approached this example by creating a deriving instance `Integral n => Integral (Maybe n)`, as `div` is part of the `Integral` type class. But this would involve implementing several *other* type classes. Explore the documentation, to determine what type classes are involved, and what functions they contain.

continues...

Exercise 3.4 Implement the three functions `lookupWithError`, `lookupWithDefault`, and `lookupInDictionary` by direct recursions, i.e., without calling `lookup`.

Exercise 3.5 A common data structure is a *rose* tree. This is a kind of tree in which each node holds a value of a particular type. The actual declarations are a bit different (they rely on Haskell's *record* syntax, which we'll see in due course), but they amount to:

```
-- | A rose tree.

data Tree a = Node a (Forest a)
type Forest a = [Tree a]
```

Note that recursion can be *mutual*, and need not be direct.

A tree consists of a node, which has two constituents: the value of type `a`, and a list of children.

Rose trees are often used to represent semi-structured data, e.g., an outline, or an XML infoset.

Write a function `preorder :: Tree a -> [a]` which returns the values contained in a `Tree` as a list, based on a preorder traversal (i.e., the value at a node comes before the values at its children). It may be helpful to know about the function `concat :: [[a]] -> [a]`, which flattens a list of lists into a simple list. (Note that the actual type of `concat` is just a bit more general than this.)

Last modified: September 07, 2020