Haskell
Lecture 7

Defining Types

type

- *type MyType = SomeType*
- *MyType is not a new type, it is another name for SomeType, a synonym*
- *see type.hs*

Type Classes

- *Operators such as == are used with different types*
- *Instead of defining a different operator for equality for each type, a type class Eq is defined for types that have equality, including Bool, Int, Char, Double, etc.*
- *Look at :info elem in ghci to see Eq a => a ...*
- *Look at :info Eq, :info Num, etc.*
- *We are not writing type classes or instances*

Define a new Data Type

- *data MyDataType = expression*
  - *data BookInfo = Book String [String]*
- *Here, Book is a data constructor. Often coders use the same name for data type and constructor*

Enumerated Type

- *data types which have constructors which take no arguments. examples*
- *data Move = Rock | Paper | Scissors*
- *data Season = Winter | Spring | Summer | Fall*

Product Types

- *data types with a single constructor*
  - *BookInfo already defined*
  - *data People = Person Name Age*
  - *type Name = String*
  - *type Age = Int*

Sum Types

- *have a number of constructors taking different arguments.*

- ```
  data Billing = CreditCard Int
               | PayPal String
               deriving (Show)
  ```

- *deriving allows making data type an instance of a type class*
  ```

Getting data from a variable of a data type

- *Use pattern matching*
- *Use named fields*
- *Examples in dataType.hs*

Recursive Data Types

- *A data type may have its name in the definition*
- *For example, a tree node for a binary search tree:*
  - *data BSTnode = MTnode |*
    Node BSTnode Int BSTnode
- *see BinSrchTree.hs for data type and insert function*

Parameterized Data Types

- *The defined data type takes a parameter*
- *Maybe, used for errors, is an example: definition*
  - *data Maybe a = Just a | Nothing*
- *see maybe.hs*

Union Type Either

- *Either takes two parameters*
  - *data Either a b = Left a | Right b*
    deriving (Eq,Ord,Read,Show)
- *import Data.Either*
- *use isLeft, isRight, fromLeft, fromRight*

Either

- *By convention, Right is the right answer and Left is the wrong answer*
- *Example showing division (where denominator of 0 is wrong) from "Real World Haskell"*
- *see either.hs*