Лекция 10. Типы данных и классов в Haskell

Пережогин А.С.

11 октября 2012 г.

Введение

Каждая переменная или функция в Haskell обязательно имеет тип. При вводе чисел или других выражений Haskell самостоятельно определяет тип вводимых данных.
Пример.

```
letter :: Char
factorial :: Int -> Int
```

letter имеет тип Char
factorial имеет тип Int -> Int

Классы типов

Каждый тип может быть включен в определенный класс.
Принадлежность тому или иному классу дает возможность выполнять действия, которые предопределены в данном классе.
Пример. Тип функции (==)

```
> :t (==)
(==) :: (Eq a) => a -> a -> Bool
```

Данная функция принимает 2 аргумента *a* и возвращает значение Bool. При этом *a* должен принадлежать классу Eq.
Например, тип Int принадлежит классу Eq.

Основные классы

- Eq – класс используется для сравнения входных выражений.

```
> 5 == 5
True
> 5 /= 5
False
```

- Ord – класс, к которому принадлежат упорядоченные данные. Функции, принадлежащие данному классу: >,<,<=,>=.
- Show – представляет класс, который преобразуется в тип String. Функция данного класс show – отображает строку.
- Read – класс противоположный по смыслу к Show. read – считывает из строки тип данных.
- Enum – перечислимый класс. Объекты этого класса . Функции pred, succ используются для нахождения следующего элемента и предыдущего.
- Num – числовой класс. Объекты этого класса ведут себя как числа.
- ЗАМЕЧАНИЕ: классы можно понимать как интерфейс в Java, но не как классы в Java.

Пользовательские типы данных

Зарезервированное слово **data** используется для определения типа.
Например. Тип Bool.

```
data Bool = False | True
```

После знака равенства идут имена конструкторов: False, True.

```
data Int = -2147483648 | -2147483647 |  ... |-1|0|1| ... | 2147483647
data Shape = Circle Float Float Float | Square Float
```

Каждый созданный тип снова является функцией.
Добавим созданный пользовательский тип Shape в класс Show

```
data Shape = Circle Float Float Float | Square Float deriving (Show)
```

Теперь можно выполнить команду

```
> Circle 5 4 4
Circle 5.0 4.0 4.0
> map (Circle 10 20) [4,5,6]
[Circle 10.0 20.0 4.0,Circle 10.0 20.0 5.0,Circle 10.0 20.0 6.0]
```

Пользовательские типы данных

Возможно создание параметризованного типа данных
Пример создания типа данных Вектор с параметром a. То есть a может иметь любой тип данных, принадлежащий к числовому классу.

```
data Vector a = Vector a a a deriving (Show)

vplus :: (Num t) => Vector t -> Vector t -> Vector t
(Vector i j k) `vplus` (Vector l m n) = Vector (i+l) (j+m) (k+n)

vectMult :: (Num t) => Vector t -> t -> Vector t
(Vector i j k) `vectMult` m = Vector (i*m) (j*m) (k*m)

scalarMult :: (Num t) => Vector t -> Vector t -> t
(Vector i j k) `scalarMult` (Vector l m n) = i*l + j*m + k*n
```

Конструкторы типов данных можно дополнить именами полей:

Пользовательские типы данных

```
data Car = Car { company :: String
               , model :: String
               , year :: Int
               } deriving (Show)
```

Пример параметризации класса Car

```
data Car a b c = Car { company :: a
                     , model :: b
                     , year :: c
                     } deriving (Show)
```

Рекурсивные типы данных

При создании типов данных можно применять рекурсивное объявление внутри конструкторов

```
data Tree a = EmptyTree | Node a (Tree a) (Tree a)
              deriving (Show, Read, Eq)
```

Пример создания функций создания и вставки элемента дерева.

```
singleton :: a -> Tree a
singleton x = Node x EmptyTree EmptyTree

treeInsert :: (Ord a) => a -> Tree a -> Tree a
treeInsert x EmptyTree = singleton x
treeInsert x (Node a left right)
        | x == a = Node x left right
        | x < a  = Node a (treeInsert x left) right
        | x > a  = Node a left (treeInsert x right)
```

Создание класса данных и реализация конкретного типа

Приведен пример создания реализации класса Eq для типа данных TrafficLight. Описание класса Eq взято прямо из библиотеки языка Haskell.

```
class Eq a where
     (==) :: a -> a -> Bool
     (/=) :: a -> a -> Bool
     x == y = not (x /= y)
     x /= y = not (x == y)

data TrafficLight = Red | Yellow | Green

instance Eq TrafficLight where
        Red == Red = True
        Green == Green = True
        Yellow == Yellow = True
        _ == _ = False
```

Контрольные задания

- Создать тип Геометрические Фигуры: Квадрат, Окружность, Треугольник.
- Создать функции вычисления площади, периметра для каждой фигуры.

- Создать тип Библиотека: Книга, Журнал, Газета.
- Создать функции определения года издания газеты, журнала, книги. Создать функцию выбора журналов из библиотечного списка.