Лекция 3. Программирование на языке Haskell

Пережогин А.С.

22 февраля 2012 г.

Типы данных

- Тип данных – совокупность однородных значений.
- Например, базовый тип – Bool
- Тип Bool включает в себя только 2 значения

  `False, True`

Тип Error

- Если применяется функция с двум или нескольким аргументам, которая не может выполнена с данным набором значений, то возвращается тип ERROR.
- Пример ошибки интерпретатора Haskell

```
>1+False
ERROR
```

- 1 – является целочисленным значением Integer, False – является переменной типа Bool
- + – определена только для двух целых чисел
- Результат выполнения – ERROR

Типы в Haskell

- При вычислении функций можно задавать напрямую тип значения, которое будет возвращено в результате выполнения
- expression :: type
- Каждое правильно построенное выражение имеет тип, который вычисляется автоматически во время компиляции на стадии проверки типов
- Все ошибки выявляются на стадии компиляции, вследствии того, что язык Haskell является строго типизированным языком.
- Команда :type выражение – возвращает тип выражения без его вычисления
- ```
  >not False
  True
  ```
- ```
  >:type not False
  not False :: Bool
  ```

Типы в Haskell

- Базовые типы в Haskell
  Bool – логические значения
  Char – символ
  String – список символов
  Int – целочисленный тип фиксированного размера
  Integer – целочисленный тип произвольного размера
  Float – числа с плавающей точкой

Типы списков в Haskell

- Список – последовательность значений одного типа
- Список логических значений

  ```
  [False, True, False] :: [Bool]
  ```

- Список символов

  ```
  ['a','b'] :: [Char]
  ```

- В общем случае, [t] – тип списка с элементами типа t
- Тип списка не связан с его длинной:

  ```
  [False, True] :: [Bool]
  [False, True, False] :: [Bool]
  ```

- В качестве элементов списка могут выступать списки:

  ```
  [['a'],['b','c']] :: [[Char]]
  ```

Кортежи в Haskell

- Кортеж – последовательность значений различного типа
- Для объявления кортежа используются круглые скобки:

```
(False, True) :: (Bool, Bool)
(False, 'a', 5) :: (Bool, Char, Integer)
```

- В общем случае, кортеж (t1, t2, .. , tn) – n-местный кортеж, компоненты которого имеют любые типы данных
- Тип кортежа зависит от его размера
  Следующие кортежи являются различными

```
(False,True)       :: (Bool,Bool)
(False,True,False) :: (Bool,Bool,Bool)
```

- Тип компонентов кортежа неограничен:

```
('a',(False,'b')) :: (Char,(Bool,Char))
(True,['a','b'])  :: (Bool,[Char])
```

Типы функций в Haskell

- Функция – правило, которое по заданному типу значения ставить в соответствие значение того же или другого типа.
- Пример. Функция not

```
not :: Bool -> Bool
```

- Пример. Функция isDigit

```
isDigit :: Char -> Bool
```

- В общем случае, t1 -> t2 – тип функции, которая переводит значения из t1 в t2
- Тип аргументов и возвращаемое значение не ограничены. Например функция может получать на вход список, кортеж и возвращать список, кортеж или отдельное значение.
- Пример 1.

```
add :: (Int,Int) -> Int
add (x,y)  = x+y
```

- Пример 2.

```
zeroto :: Int -> [Int]
zeroto n = [0..n]
```

Каррирование в Haskell

- Функции нескольких переменных рассматриваются в Haskell могут возвращать в качестве значения функции и далее применять эти функции к оставшимся аргументам

```
add :: Integer -> (Integer -> Integer)
add x y = x+y
```

- Сравнить.

```
add2 :: (Integer, Integer) -> Integer
add2 (x,y) = x+y
```

- Каррирование (Haskell Curry) – операция ассоциативная влево, результатом может быть значение или функция, котором потом применяется к следующему аргументу.
- Функция трех переменных x,y,z

```
mult :: Int -> (Int -> (Int -> Int))
mult x y z = x*y*z
:type mult 3 5 :: Integer -> Integer
```

Каррирование в Haskell

- Для удобства упрощения записи функций, использующих операцию каррирования, опускают лишние скобки. При этом понимают следующую запись

  ```
  Integer -> Integer -> Integer -> Integer
  ```

  как

  ```
  Integer -> (Integer -> (Integer -> Integer))
  ```

Полиморфные функции в Haskell

- Функция называется полиморфной, если она может принимать в качестве аргумента более одного типа аргументов.
- Пример.

  `length :: [a] -> Int`

  функция принимает список произвольных значений и возвращает число элементов списка
- Переменная a может иметь любой тип:

```
length [False, True] -- a = Bool
length [1,2,3,4] -- a = Integer
```

- Типовые переменные обозначают прописными буквами и чаще всего a, b, c
- Многие функции в файле Prelude.hs являются полиморфными.

```
first :: (a,b) -> a
head :: [a] - > a
id :: a -> a
```

Перегруженные полиморфные функции в Haskell

- Полиморфная функция является перегруженной, если она ее аргументы принадлежать к определенному классу.

  ```
  sum :: Num a => [a] -> a
  ```

  Тип переменной a в функции sum должен быть числовым
- В Haskell определены 3 базовых класса:
  - Num – числовой тип
    Например,

    ```
    (+) :: Num a => a -> a -> a
    ```
  - Eq – сравнительный тип

    ```
    (==) :: Eq a => a -> a -> Bool
    ```
  - Ord – упорядоченный тип

    ```
    (<) :: Ord a => a -> a -> Bool
    ```

Замечания

- При задании функции рекомендуется указать явно её тип
- Для полиморфных функций, которые используют переменные из классов Num, Eq, Ord, необходимо включить принадлежность к определенному классу

Контрольные задания

1. Какой тип имеют следующие выражения:

```
['a','b','c']
('a','b','c')
[(5,4,False),(3,7,True)]
([5,4,False],[3,7])
```

2. Указать какой тип имеют следующие функции

```
second xs      = head (tail xs)
swap (x,y)     = (y,x)
pair x y       = (x,y)
double x       = x*2
palindrome xs = reverse xs == xs
twice f x      = f (f x)
```

3. Проверить с помощью hugs