Лекция 4. Объявление функций в Haskell

Пережогин А.С.

2 апреля 2012 г.

Выражения с условием

- Как и во многих языках программирования функции могут быть определены с использованием условного оператора

```
abs :: Int -> Int
abs n = if n >= 0 then n else -n
```

- Условный оператор может быть вложенным

```
signum :: Int -> Int
signum n = if n < 0 then -1 else if n == 0 then 0 else 1
```

- Условный оператор обязательно имеет if then else. Это позволяет избежать неоднозначность.

Условное задание фукнции

- В качестве альтернативы условному оператору функции можно определять используя кусочное задание функции.

```
abs n | n >= 0 = n
      | otherwise = -n
```

- Позволяет удобно определять кусочные функции
- otherwise определено как true

Сопоставление с образцом

- Многие функции имеют специальное определение через сопоставление с образцом для их аргументов

```
not :: Bool -> Bool
not False = True
not True  = False
```

- Используем сопоставления с образцом для функции конъюнкции:

```
(&&) :: Bool -> Bool -> Bool
True  && True  = True
True  && False = False
False && True  = False
False && False = False
```

- В компактном виде функцию можно объявить с помощью _

```
(&&) :: Bool -> Bool -> Bool
True && True  = True
_    && _     = False
```

Сопоставление с образцом

- Символ _ соответствует любому значению аргумента заданного типа
- Сопоставление с образцом зависит от порядка появления объявления функции

```
_ && _ = False
True && True = True
```

Всегда будет возвращать значение False

- При сопоставлении с образцом переменные не повторяются. Переменная каждого аргумента уникальна.

```
b && b = b
_ && _ = True
```

Результат – ошибка выполнения

Выделение образца в списках

- Каждый список задан с помощью оператора **:**, который добавляет элемент в список

  `[1,2,3,4]`

  С точки зрения оператора означает 1:(2:(3:(4:[]))), где [] – пустой список
- Функции работы со списком head и tail определяются с помощью образца x:xs

```
head  ::[a] -> a
head (x:xs) = x

tail  ::[a] -> [a]
tail (x:xs) = xs
```

- Аргумент x:xs должен быть не пустой.
- Функция имеет более высокий приоритет по сравнению с :, поэтому обязательно заключать в скобки (x:xs).

Выделение образца для целых чисел

- В качестве образца для целых чисел можно использовать представление числа в виде (n+k), где $k > 0$

```
pred :: Int -> Int
pred (n+1) -> n
```

Возвращает предыдущее числе перед n

- В образце (n+k) $k > 0$

```
pred 0
Error
```

- Образец обязательно заключается в скобки (n+k), следуя приоритету операций

Лямбда-выражения

- Функцию можно создавать без присвоения имени функции с помощью лямбда-выражений

```
Main> (\x -> x + x) 5
10
```

- Лямбда-выражение обозначается символом λ. Например, $\lambda x.(x + x)$. В Haskell используется символ \.
- Лямбда-выражения позволяют удобно записать функции, которые используют операцию каррирования.

```
add x y = x+y
add = \x -> (\y -> x+y)
```

- Лямбда-выражения удобно использовать, когда результатом работы функции является функция

```
const    :: a -> b -> a
const x _ = x

const    :: a -> (b -> a)
const x = \_ -> x
```

Лямбда-выражения

- Удобно использовать без объявления имени функции

  I способ

```
odds n = map f [0..n-1]
         where
            f x = x*2 + 1
```

  II способ

```
odds n = map (\ x -> x*2 + 1) [0..n-1]
```

Сечения (sections)

- Оператор между двумя операндами может быть преобразован в запись функции с помощью операции каррирования

```
1+2
(+) 1 2
```

- Такое преобразование позволяет добавлять только одни элемент

```
(1+) 2
(+2) 1
```

- Такая запись функций позволяет более удобно записывать разные функции:

```
(1+) -- функция увеличения значения на 1
(1/) -- функция вычисления дроби 1/
(*2) -- функция удвоения
(/2) -- функция деления пополам
```

Контрольные задания

- Создать функцию safetail, которая работает аналогично tail, но при передаче пустого списка возвращает пустой список.
  Использовать: условное выражение, охраняемые условия, сопоставление по образцу.
  Замечание. Библиотечная функция проверяет пустой список или нет.

  ```
  null :: [a] -> Bool
  ```

- Объявить 3 логические функции дизъюнкция, импликация, XOR.