CS314. Функциональное программирование

Введение в программирование на языке Haskell

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

Содержание

1. Среда программирования
2. Операции и функции
3. Списки и генераторы списков
4. Кортежи
5. Пример: квадратные уравнения
6. Импорт модулей
7. Написание собственных модулей

Порядок работы

- Интерпретатор GHCi.
- Текстовый редактор с подсветкой синтаксиса + терминал.
- Команды GHCi:
  - `:load <имя файла>` или `:l <имя файла>`
  - `:reload` или `:r`
  - `:quit` или `:q`

Примеры

Факториал числа (fact.hs)

```
factorial 0 = 1
factorial n = n * factorial (n-1)
```

«Привет, мир» (hello.hs)

```
main = putStrLn "Привет, мир"
```

Содержание

1 Среда программирования

2 Операции и функции

3 Списки и генераторы списков

4 Кортежи

5 Пример: квадратные уравнения

6 Импорт модулей

7 Написание собственных модулей

Операции

```
ghci> 2 + 3
5
ghci> 3 - 8
-5
ghci> 3 * 8
24
ghci> 3 / 5
0.6
ghci> 2 ^ 5
32
ghci> 64 ** 0.5
8.0
ghci> 2 + 3 * (-6)
-16
```

```
ghci> 1 < 3
True
ghci> 5 >= 10
False
ghci> 3 == 3
True
ghci> 5 /= 5
False
ghci> not True
False
ghci> True && False
False
ghci> False || True
True
```

Вызов функций

```
ghci> succ 8
9
ghci> min 2 8
2
ghci> max 0.6 2.3
2.3
ghci> succ 9 + max 5 4 + 1
16
ghci> (succ 9) + (max 5 4) + 1
16
ghci> div 92 10
9
ghci> 92 `div` 10
9
ghci> 92 `mod` 10
2
```

Объявление функций

doubles.hs

```
doubleMe x = x + x

doubleUs x y = doubleMe x + doubleMe y
```

GHCi

```
ghci> doubleMe 5
10
ghci> doubleUs 2 3
10
```

Проверка условий

Условное выражение

```
lucky x = if x == 7
            then "СЧАСТЛИВОЕ ЧИСЛО 7!"
            else "Прости, друг, повезёт в другой раз!"

f a = (if a < 0 then (-a) else a) + 1
```

Проверка условий

Охранные выражения (guards)

```
lucky x
  | x == 7 = "СЧАСТЛИВОЕ ЧИСЛО 7!"
  | otherwise = "Прости, друг, повезёт в другой раз!"

f a
  | a < 0 = (-a) + 1
  | a >= 0 = a + 1
```

Сопоставление с образцом (pattern matching)

```
lucky 7 = "СЧАСТЛИВОЕ ЧИСЛО 7!"
lucky x = "Прости, друг, повезёт в другой раз!"

factorial 0 = 1
factorial n = n * factorial (n-1)

sayMe 1 = "Один!"
sayMe 2 = "Два!"
sayMe 3 = "Три!"
sayMe 4 = "Четыре!"
sayMe 5 = "Пять!"
sayMe n = "Это число не в пределах от 1 до 5"
```

Образцы и охранные выражения

Анализ индекса массы тела

```
bmiTell _ 0 = "Мелких не обслуживаем!"
bmiTell weight height
  | weight / height ^ 2 <= 18.5 = "Слышь, эмо, ты дистрофик!"
  | weight / height ^ 2 <= 25.0 = "По части веса ты в норме.
                                   Зато, небось, уродец!"
  | weight / height ^ 2 <= 30.0 = "Ты толстый!
                                   Сбрось хоть немного веса!"
  | otherwise = "Мои поздравления, ты жирный боров!"
```

Конструкция where

Анализ индекса массы тела

```
bmiTell _ 0 = "Мелких не обслуживаем!"
bmiTell weight height
    | bmi <= skinny = "Слышь, эмо, ты дистрофик!"
    | bmi <= normal = "По части веса ты в норме.
                       Зато, небось, уродец!"
    | bmi <= fat = "Ты толстый! Сбрось хоть немного веса!"
    | otherwise = "Мои поздравления, ты жирный боров!"
  where
    bmi = weight / height ^ 2
    skinny = 18.5
    normal = 25.0
    fat = 30.0
```

Выражение let

Площадь поверхности цилиндра

```
cylinder r h =
  let
    sideArea = 2 * pi * r * h
    topArea = pi * r ^ 2
  in
    sideArea + 2 * topArea
```

```
ghci> 4 * (let a = 9 in a + 1) + 2
42
```

Конструкция let в GHCi

```
ghci> let a = 10
ghci> a
10
ghci> a + 10
20
ghci> let f n = n * n
ghci> f 5
25
ghci> let g n = n ^ n in g 5
3125
ghci> g 2

<interactive>:1:1: Not in scope: `g'
```

Объявление функций в let и where

```
g a = f (a+1) + f (a−1)
  where
    f 0 = 0
    f n = 2*n + 1
```

```
g a =
  let
    f 0 = 0
    f n = 2*n + 1
  in f (a+1) + f (a−1)
```

Отступы в коде программы

Основные правила (неформально)

- Код, являющийся частью выражения, должен находиться правее начала этого выражения.
- Выражения, составляющие единую группу, должны находиться на одном уровне.

```
g a =
  let
    f 0 = 0
    f n = 2*n + 1
  in f (a+1) + f (a-1)
```

- Если что-то не работает (bad layout) — проверьте отступы.

Явное задание типа функции

```
cylinder :: Double -> Double -> Double
cylinder r h =
  let
    sideArea = 2 * pi * r * h
    topArea = pi * r ^ 2
  in
    sideArea + 2 * topArea
```

Содержание

1 Среда программирования

2 Операции и функции

3 Списки и генераторы списков

4 Кортежи

5 Пример: квадратные уравнения

6 Импорт модулей

7 Написание собственных модулей

Списки и операции с ними

- Список — гомогенная структура данных.

```
ghci> let lostNumbers = [4,8,15,16,23,42]
ghci> lostNumbers
[4,8,15,16,23,42]
ghci> [1,2,3,4] ++ [9,10,11,12]
[1,2,3,4,9,10,11,12]
ghci> "hello," ++ " " ++ "world"
"hello, world"
ghci> ['a','b'] ++ ['c','d']
"abcd"
ghci> :t "abcd"
"abcd" :: [Char]
```

Пустой список и операция конструирования списка (:)

```
ghci> 5 : [ ]
[5]
ghci> 1 : 2 : 3 : [ ]
[1,2,3]
ghci> 1 : [2,3]
[1,2,3]
ghci> :t [ ]
[ ] :: [a]
ghci> :t [5]
[5] :: Num t => [t]
ghci> :t [5,10]
[5,10] :: Num t => [t]
```

Простейшие функции обработки списков

```
ghci> head [5,4,3,2,1]
5
ghci> tail [5,4,3,2,1]
[4,3,2,1]
ghci> last [5,4,3,2,1]
1
ghci> init [5,4,3,2,1]
[5,4,3,2]
ghci> head [ ]
*** Exception: Prelude.head: empty list
ghci> length [5,4,3,2,1]
5
```

Простейшие функции обработки списков

```
ghci> take 3 [5,4,3,2,1]
[5,4,3]
ghci> take 1 [3,9,3]
[3]
ghci> take 5 [1,2]
[1,2]
ghci> take 0 [6,6,6]
[]
ghci> drop 3 [8,4,2,1,5,6]
[1,5,6]
ghci> drop 0 [1,2,3,4]
[1,2,3,4]
ghci> drop 100 [1,2,3,4]
[]
```

Простейшие функции обработки списков

```
ghci> null [1,2,3]
False
ghci> null [ ]
True
ghci> reverse [5,4,3,2,1]
[1,2,3,4,5]
ghci> minimum [8,4,2,1,5,6]
1
ghci> maximum [1,9,2,3,4]
9
ghci> sum [5,2,1,6,3,2,5,7]
31
ghci> product [6,2,1,2]
24
```

Списки: индексация и проверка принадлежности

```
ghci> "ABCDEF" !! 2
'C'
ghci> [9.4,33.2,96.2,11.2,23.25] !! 1
33.2
ghci> 4 `elem` [3,4,5,6]
True
ghci> 10 `elem` [3,4,5,6]
False
```

Интервалы

```
ghci> [1..20]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
ghci> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
ghci> ['K'..'Z']
"KLMNOPQRSTUVWXYZ"
ghci> [2,4..20]
[2,4,6,8,10,12,14,16,18,20]
ghci> [3,6..20]
[3,6,9,12,15,18]
ghci> [5,4..1]
[5,4,3,2,1]
```

Бесконечные списки

```
ghci> take 10 [1..]
[1,2,3,4,5,6,7,8,9,10]
ghci> take 10 (repeat 5)
[5,5,5,5,5,5,5,5,5,5]
ghci> take 10 (cycle [1,2,3])
[1,2,3,1,2,3,1,2,3,1]
ghci> take 11 (cycle "LOL ")
"LOL LOL LOL"
```

Генераторы списков

```
ghci> [x*2 | x <- [1..10]]
[2,4,6,8,10,12,14,16,18,20]

ghci> [x*2 | x <- [1..10], x*2 >= 12]
[12,14,16,18,20]
```

«Бум и Бах»

```
boomBangs xs = [if x < 10 then "БУМ!" else "БАХ!" | x <- xs, odd x]

ghci> boomBangs [7..13]
["БУМ!","БУМ!","БАХ!","БАХ!"]
```

Генераторы списков с несколькими источниками

```
ghci> [x*y | x <- [2,5,10], y <- [8,10,11]]
[16,20,22,40,50,55,80,100,110]

ghci> [x*y | x <- [2,5,10], y <- [8,10,11], x*y > 50]
[55,80,100,110]
```

Сопоставление с образцом для списков

Суммирование списка целых

```
sum' [ ] = 0
sum' (x:xs) = x + sum' xs
```

```
ghci> sum' [ ]
0
ghci> sum' [1..10]
55
```

Безопасная head

```
head' def [ ] = def
head' _   (x:_) = x
```

Сопоставление с образцом для списков

Анализ списка

```
tell :: (Show a) => [a] -> String
tell [ ] = "Список пуст"
tell [x] = "В списке один элемент: " ++ show x
tell [x,y] = "В списке два элемента: " ++ show x ++ " и " ++ show y
tell (x:y:_) = "Список длинный. Первые два элемента: " ++ show x
                 ++ " и " ++ show y
```

```
ghci> tell [True, False]
"В списке два элемента: True и False"
ghci> tell [1, 2, 3, 4]
"Список длинный. Первые два элемента: 1 и 2"
```

Содержание

1 Среда программирования

2 Операции и функции

3 Списки и генераторы списков

4 Кортежи

5 Пример: квадратные уравнения

6 Импорт модулей

7 Написание собственных модулей

Кортежи

- Кортеж — гетерогенная структура данных.
- Кортеж с двумя элементами называется парой.

```
ghci> (1, 3)
(1,3)
ghci> (3, 'a', "hello")
(3,'a',"hello")
ghci> (50, 50.4, "hello", 'b')
(50,50.4,"hello",'b')
ghci> :t (1, 'a', True)
(1, 'a', True) :: Num t => (t, Char, Bool)
```

Функции для пар

```
ghci> fst (8,11)
8
ghci> fst ("Bay", False)
"Bay"
ghci> snd (8,11)
11
ghci> snd ("Bay", False)
False
ghci> :t fst
fst :: (a, b) -> a
```

Сопоставление с образцом для кортежей

Сложение векторов

```
addVectors :: (Double, Double)
                -> (Double, Double) -> (Double, Double)
addVectors (x1, y1) (x2, y2) = (x1 + x2, y1 + y2)
```

```
ghci> addVectors (1,2) (3,4)
(4.0,6.0)
```

Содержание

1 Среда программирования

2 Операции и функции

3 Списки и генераторы списков

4 Кортежи

5 Пример: квадратные уравнения

6 Импорт модулей

7 Написание собственных модулей

Решение квадратного уравнения

```
solveSqEq :: Double -> Double -> Double -> [Double]
solveSqEq 0 _ _ = error "Это не квадратное уравнение"
solveSqEq a b c = findRoots discr
  where
    discr = b*b - 4*a*c
    a2 = 2 * a
    findRoots d
      | d < 0 = [ ]              -- корней нет
      | d == 0 = [-b/a2]         -- один корень
      | otherwise =              -- два корня
          let
            m1 = -b / a2
            m2 = sqrt d / a2
          in  [m1 - m2, m1 + m2]
```

Решение набора квадратных уравнений

```
{-
   Решение набора квадратных уравнений.
   Набор задаётся списком троек — коэффициентов уравнений.
-}
solveSqEqs :: [(Double, Double, Double)] -> [[Double]]
solveSqEqs [ ] = [ ]
solveSqEqs ((a,b,c):koeffs) = solveSqEq a b c : solveSqEqs koeffs
```

```
ghci> solveSqEq 1 (−5) 6
[2.0,3.0]
ghci> solveSqEqs [(1,2,5),(1,2,1),(1,−8,15)]
[[ ],[−1.0],[3.0,5.0]]
```

Содержание

1 Среда программирования

2 Операции и функции

3 Списки и генераторы списков

4 Кортежи

5 Пример: квадратные уравнения

6 Импорт модулей

7 Написание собственных модулей

Импорт всего содержимого модуля

```
import Data.List
import Data.Char

numUniques :: (Eq a) => [a] -> Int
numUniques = length . nub

numLetters :: [Char] -> Int
numLetters = length . filter isAlpha
```

Импорт модуля: варианты

Избирательный импорт

import Data.**List** (**nub**, **sort**)

- Импортируются только указанные функции.

Скрывающий импорт

import Data.**List** **hiding** (**nub**, **group**)
import **Prelude** **hiding** (**map**)

- Импортируется всё, кроме указанных функций.

Квалифицированный импорт

```
import qualified Data.List

numUniques :: (Eq a) => [a] -> Int
numUniques = length . Data.List.nub
```

```
import qualified Data.List as L

numUniques :: (Eq a) => [a] -> Int
numUniques = length . L.nub
```

- Не путайте композицию функций и квалифицированное имя!
- Часто используется при совпадении имён в разных модулях.

Информация о модулях

- Модули организованы в библиотеки (стандартные и внешние).
- Список основных модулей:
  `http://www.haskell.org/ghc/docs/latest/html/libraries/`.
- Hoogλe — поиск по API: `http://www.haskell.org/hoogle/`.
  Примеры поисковых запросов:
  - map
  - (a -> b) -> [a] -> [b]
  - Ord a => [a] -> [a]
  - Data.Map.insert
- Душкин Р. В. Справочник по языку Haskell. М.: ДМК Пресс, 2008.

Содержание

1 Среда программирования

2 Операции и функции

3 Списки и генераторы списков

4 Кортежи

5 Пример: квадратные уравнения

6 Импорт модулей

7 Написание собственных модулей

Простейший модуль и его использование

Файл ModuleName.hs

```
module ModuleName (funA, funB) where

funA  = ...

funB = ...

-- Вспомогательная функция, не экспортируется
funAux = ...
```

- Имя модуля должно начинаться с большой буквы.
- Имя файла должно совпадать с именем модуля и иметь расширение .hs.
- После имени модуля идёт список экспортируемых функций.

Использование простейшего модуля

```
import ModuleName

{- Использование функций, экспортируемых модулем ModuleName -}
```

- Файл с модулем должен находиться в том же каталоге.

Проект Geometry

Задача

Требуется вычислять объёмы и площади поверхностей сферы, прямоугольного параллелепипеда и куба.

Разбиение на модули

- Модуль Geometry.Sphere (файл Geometry/Sphere.hs).
- Модуль Geometry.Cuboid (файл Geometry/Cuboid.hs).
- Модуль Geometry.Cube (файл Geometry/Cube.hs).
- Все файлы модулей находятся в каталоге Geometry!

Модуль Geometry.Sphere

```
module Geometry.Sphere
( volume
, area
) where

volume :: Float -> Float
volume radius = (4.0 / 3.0) * pi * (radius ^ 3)

area :: Float -> Float
area radius = 4 * pi * (radius ^ 2)
```

Модуль Geometry.Cuboid

```
module Geometry.Cuboid
( volume
, area
) where

volume :: Float -> Float -> Float -> Float
volume a b c = rectArea a b * c

area :: Float -> Float -> Float -> Float
area a b c = rectArea a b * 2 + rectArea a c * 2 + rectArea c b * 2

rectArea :: Float -> Float -> Float
rectArea a b = a * b
```

Модуль Geometry.Cube

- Куб — это прямоугольный параллелепипед, все рёбра которого равны.

```
module Geometry.Cube
( volume
, area
) where

import qualified Geometry.Cuboid as Cuboid

volume :: Float -> Float
volume side = Cuboid.volume side side side

area :: Float -> Float
area side = Cuboid.area side side side
```

Использование модулей

- Файл, импортирующий модуль, должен находиться на одном уровне с каталогом Geometry.

Импорт одного модуля

```
import Geometry.Sphere

... area ...
```

Импорт нескольких модулей

```
import qualified Geometry.Sphere as Sphere
import qualified Geometry.Cuboid as Cuboid
import qualified Geometry.Cube as Cube

... Cube.area ...
```