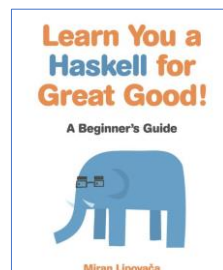


Кратък увод в Haskell

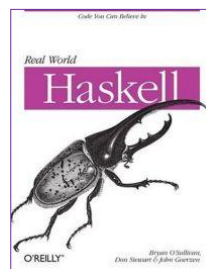
доц. Атанас Семерджиев

1

Miran Lipovaca (2011)
Learn You a Haskell for Great Good!
<http://learnyouahaskell.com/>



Bryan O'Sullivan, Don Stewart, and
John Goerzen (2008)
Real World Haskell
<http://book.realworldhaskell.org/>



2

Коментари

```
-- Това е коментар
```

```
{- Това също е коментар,  
който може да продължи и  
на следващите редове -}
```

3

3

Literate Programming / Bird Style

В този случай текстът, който пишете свободно в Source файла се счита за коментар.

Ако искате да напишете реален код, трябва да сложите пред него символа (>). Например:

```
> fact 0 = 1  
> fact x = x * fact (x - 1)
```

За да може да пишете в този стил, разширението на файла трябва да бъде .LHS

4

4

Типове

Езикът е силно типизиран (strongly typed).

Някои от поддържаните типове:

- `Int`
- `Integer`
- `Float`
- `Double`
- `Boolean` (истина и лъжа са съответно: `True` и `False`)
- `Char` (например: `'a'`, `'b'`, `'c'`)

5

5

Извикване на функция

1. Функциите се извикват без скоби!

```
> min 5 10  
5
```

```
> succ 1  
2
```

2. Двухаргументните функции могат да се извикват или префиксно, или инфиксно:

```
> 5 `min` 10  
5
```

6

6

Приоритет на операцията

Прилагането на функция е с висок приоритет.

По тази причина, следният код ще предизвика грешка:

```
negate max 10 20
```

Има различни начини да преодолеем това. Например:

```
negate (max 10 20)  
negate $ max 10 20
```

7

Приоритет на операцията

Прилагането на функция е с висок приоритет.

По тази причина, следният код ще предизвика грешка:

```
negate max 10 20
```

Има различни начини да преодолеем това. Например:

```
negate (max 10 20)  
negate $ max 10 20
```

8

Application operator (\$)

Може да се използва многократно, например:

```
> negate $ length $ words "one two three"  
-3
```

Сравнете горния запис и следния:

```
> negate (length (words "one two three"))
```

9

If-then-else

1. Else клаузата е задължителна!

```
f x = if x > 0 then x else (-x)
```

2. Както и в Scheme, if-then-else може да участва в израз:

```
(if x > 0 then x else (-x)) * 10
```

10

10

If-then-else

Разпространена конвенция в Haskell е `else` да се подравнява с `then`, а не с `if`.

```
f x = if x >= 0 && x <= 9
      then "one digit"
      else if x >= 10 && x <= 99
            then "two digits"
            else "more than two"
```

11

11

let

```
a = let x = 10
      y = 20
      in x + y
```

```
b = let x = 10; y = 20 in x + y
```

```
c = (let x = 10; y = 20 in x + y) * 2
```

За повече информация вижте: <https://en.wikibooks.org/wiki/Haskell/Indentation>

12

12

let

```
a = let x = 10  
      y = 20  
      in x + y
```

```
b = let { x = 10;  
          y = 20  
        } in x + y
```

```
-- Works, but please, never do this!  
c = let { x = 10;  
        y = 20  
      } in x + y
```

13

13

Списъци

14

Списък

1. Списъците се декларират с квадратни скоби:

```
[1, 2, 3, 4, 5]
```

2. Празният списък :

```
[]
```

3. Списъците в Хаскел са хомогенни!

```
[1, 2, 3, "Hello world!"] -- грешка!
```

15

15

Конструиране на списък

1. За конструиране на списък се използва операцията :

```
1 : [2, 3, 4, 5]
```

2. Конкатенация се извършва с операцията ++

```
[1, 2] ++ [3, 4, 5]
```

16

16

Глава и опашка

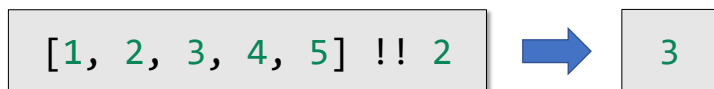
1. Глава на списък:



2. Опашка на списък:



3. Намиране на n-ти елемент:



17

17

Полезни операции със списъци

```
null []      -- True
null [2,3]   -- False
length [10,20,30] -- 3
last [1,2,3,4,5] -- 5
init [1,2,3,4,5] -- [1,2,3,4]
reverse [1,2,3,4,5] -- [5,4,3,2,1]
```

18

Полезни операции със списъци

```
take 3 [1,2,3,4,5]    -- [1,2,3]
drop 3 [1,2,3,4,5]    -- [4,5]
replicate 3 10         -- [10, 10, 10]
maximum [1,2,3,4,5]   -- 5
minimum [1,2,3,4,5]   -- 1
zip [1,2,3] [1,2,3]   -- [(1,1),(2,2),(3,3)]
unzip [(1,1),(2,2),(3,3)] -- ([1,2,3],[1,2,3])
```

19

Проверка дали елемент се съдържа в списък

```
elem 3 [1,2,3,4,5] -- True
```

Често горното се записва като:

```
3 `elem` [1,2,3,4,5]
```

20

Ranges

```
[1..10]      -- [1,2,3,4,5,6,7,8,9,10]
[1,3..10]    -- [1,3,5,7,9]
['a'..'z']
['a','c'..'z']
[10,9..1]    -- [10,9,8,7,6,5,4,3,2,1]
[10..1]      -- []
[10,9..20]   -- []
```

21

Безкрайни списъци

```
[1..]        -- [1,2,3,...]
[1,1..1]     -- [1,1,1,...]
take 4 [1,3..] -- [1,3,5,7]
repeat 5     -- [5,5,5,...]
cycle [1,2]   -- [1, 2, 1, 2, 1, 2, ...]
```

22

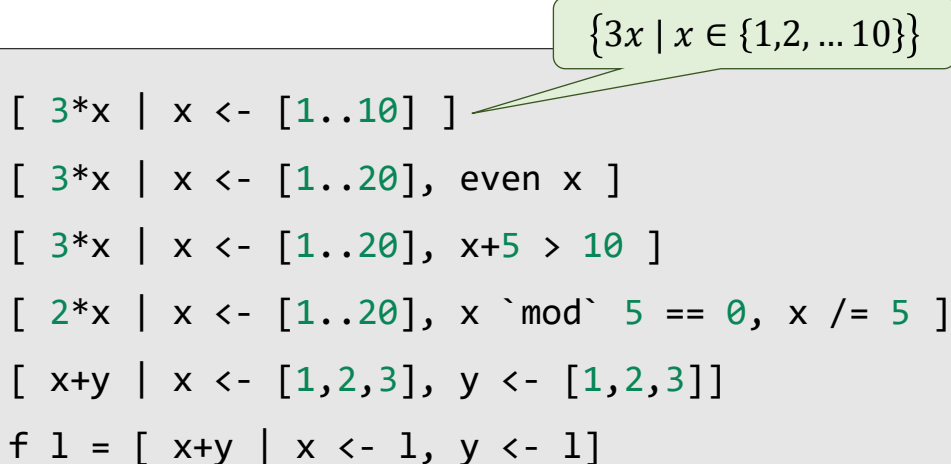
Безкрайни списъци

При работата с безкрайни списъци важат правила сходни с тези за безкрайни потоци в Scheme

```
print (take 4 [1,3..])  -- [1,3,5,7]
print [1,3..]          -- никога не приключва
```

23

List Comprehension



The diagram shows a list comprehension `[3*x | x <- [1..10]]` with a callout box pointing to it containing the mathematical set notation $\{3x \mid x \in \{1,2, \dots 10\}\}$. Below this are several other list comprehension examples.

```
[ 3*x | x <- [1..10] ]
[ 3*x | x <- [1..20], even x ]
[ 3*x | x <- [1..20], x+5 > 10 ]
[ 2*x | x <- [1..20], x `mod` 5 == 0, x /= 5 ]
[ x+y | x <- [1,2,3], y <- [1,2,3]]
f 1 = [ x+y | x <- 1, y <- 1]
```

24

Символни низове

1. Представят се като списъци:



2. Операциите със списъци се пренасят върху низовете:



3. Съществуват пакети, с които можете да работите и с други представяния. Вижте например:

<https://wiki.haskell.org/Strings>

<https://mmhaskell.com/blog/2017/5/15/untangling-haskells-strings>

25

25

Наредени n-орки

Заграждат се в кръгли скоби

```
(1,2)
(1,2,3,4)
("Abc", 1, [1,2,3])
```

Операциите са различни от тези за списъци

```
fst (1, 2)  -- 1
snd (1, 2)  -- 2
fst (1,2,3,4)  -- Грешка
snd (1,2,3,4)  -- Грешка
```

26

Деклариране на функция

Името трябва да започва с малка буква!

```
square x = x * x  
displacement v t = v * t
```

Ако работите интерактивно с интерпретатора, новите дефиниции се въвеждат с `let`:

```
let square x = x * x
```

27

27

Шаблони / Pattern matching

```
f 1 = "One"  
f 2 = "Two"  
f x = "Other"
```

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

28

28

Скобите са ключови!

```
fact 0 = 1
fact n = n * fact (n - 1)  -- ОК
```

```
fact 0 = 1
fact n = n * fact n - 1    -- безкрайно!
```

29

```
f1 _ 0 = 1
f1 x y = x / y
```

```
f3 x x = x  -- Грешка
```

```
f2 _ _ _ 0 = False
f2 _ _ 0 _ = False
f2 _ 0 _ _ = False
f2 0 _ _ _ = False
f2 _ _ _ _ = True
```

```
f4 _ _ = 5
f4 x y = x + y  -- ИЗЛИШНО
```

30

Шаблони и наредени n-орки

```
f1 a = fst a + snd a
```

```
f2 (x,y) = x + y
```

```
f3 (_,0) = 1  
f3 (x,y) = x / y
```

```
first  (a, _, _) = a  
second (_, b, _) = b  
third  (_, _, c) = c
```

31

31

Шаблони и списъци

```
length' [] = 0  
length' (x:xs) = 1 + length' xs
```

```
length' [] = 0  
length' (_:xs) = 1 + length' xs
```

32

32

Шаблони и списъци

```
first' (x:_) = x
```

```
second (_:x:_) = x
```

```
last' [x] = x  
last' (_:xs) = last' xs
```

```
sumOf3 [a,b,c] = a+b+c
```

33

33

```
describe [] = "Empty list"  
  
describe [x] = "One element - " ++ show x  
  
describe [x,y] = "Two elements - " ++  
                show x ++ " and " ++  
                show y  
  
describe [x,y,z] = "Three elements."  
  
describe _ = "Many elements"
```

34

As-pattern

```
info l@(x:xs) = "List is "  
              ++ show l  
              ++ " head is "  
              ++ show x
```

35

Guards

```
f x  
| x > 0 = "Positive"  
| x == 0 = "Zero"  
| otherwise = "Negative"
```

```
factorial n  
| n <= 0 = 1  
| n > 0 = n * factorial (n - 1)
```

36

36

where клауза

```
quadratic a b c
| d < 0 = "None"
| d == 0 = "One root"
| d > 0 = "Two roots"
where d = b*b - 4*a*c
```

```
square x = result
where result = x * x
```

37

37

```
quadratic a b c
| d < 0 = "None"
| d == 0 = "One root: " ++ show r1
| d > 0 = "Two roots: "
           ++ show r1
           ++ " and "
           ++ show r2
where d = b*b - 4*a*c
      r1 = (negate b + d) / 2*a
      r2 = (negate b - d) / 2*a
```

38

38

case

```
f 1 = "One"
f 2 = "Two"
f _ = "Something else"
```



```
f x = case x of 1 -> "One"
                2 -> "Two"
                _ -> "Something else"
```

39

39

case

Случаите трябва да:

- са еднакво подравнени;
- започват по-навътре от реда съдържащ `of`;

За запис на един ред може да се използва `;`

```
f x = case x of
  1 -> "One"
  2 -> "Two"
  _ -> "> 2"
```

```
f x = case x of 1 -> "One"
                2 -> "Two"
                _ -> "> 2"
```

```
f x = case x of 1 -> "One"; 2 -> "Two"; _ -> "> 2"
```

40

40

case guards

```
myAnd True y
  | y      = True
  | otherwise = False
myAnd _ _ = False
```



```
myAnd x y = case x of
  True | y      -> True
        | otherwise -> False
  _ -> False
```

41

41

Декларация на тип

```
fact :: Integer -> Integer
fact 0 = 1
fact n = n * fact (n - 1)
```

```
quadratic :: Double -> Double -> Double -> [Char]
quadratic a b c
  | d < 0 = "None"
  -- ... останалият код на функцията
```

42

42

```
5::Int  
5::Integer  
5::Double
```

```
a :: Integer  
a = 10
```

```
b = 10::Integer
```

```
f :: a -> a  
f x = x
```

```
> f 5  
5
```

```
> f "abc"  
"abc"
```

43

43

```
f :: a -> a  
f x = x
```

```
> f 5  
5
```

```
> f "abc"  
"abc"
```

```
g :: a -> [a]  
g x = [x,x]
```

```
h :: a -> (a,a)  
h x = (x, x)
```

```
j :: a -> (a,a)  
j x = (x, 5) -- грешка
```

44

44

```
j :: a -> (a,a)  
j x = (x, 5) -- грешка
```

```
j :: (Num a) => a -> (a,a)  
j x = (x, 5) -- OK
```

45

45



46