

# Úvod, opakování, pokročilejší syntax; moduly, typové třídy

IB016 Seminář z funkcionálního programování

Vladimír Štill, Martin Ukrop

Fakulta informatiky, Masarykova univerzita

Jaro 2016

## Interpret a kompilátor

- výhradně GHC(i), verze alespoň 7.8
  - pozor na změny v základní knihovně od verze 7.10
  - `module add ghc-7.8.4`
- doporučujeme použít Haskell Platform
- interpret `ghci`
  - `ghci <soubor>` spuštění
  - `:q` opuštění interpretu
  - `:t <vyraz>` otypování výrazu
  - `:i <vyraz>` informace k výrazu (asociativita a priority operátoru, definice typu)
  - `:l <soubor>` načtení souboru
- kompilátor `ghc`
  - `ghc --make <soubor>`, soubor musí obsahovat funkci `main`

# Opakování

- typ nepovinný, silně doporučený
  - pište ho dříve než tělo funkce
- definice může obsahovat více rovnic s různými vzory
- parametry odděleny mezerou

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

- Polymorfismus, seznamy, vzory

```
look :: Eq a => [(a,b)] -> a -> Maybe b
look [] _ = Nothing
look ((k,v) : s) x = if k == x then Just v
                      else look s x
```

## Opakování: funkce vyšších řádů, lambda funkce

```
map    :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
foldr  :: (a -> b -> b) -> b -> [a] -> b
foldl  :: (b -> a -> b) -> b -> [a] -> b
```

```
suma list = foldl (+) 0 list
fact n = foldl (*) 1 [2..n]
```

```
look :: Eq a => [(a,b)] -> a -> Maybe b
look list x = foldr
  (\(k, v) rest -> if k == x then Just v else rest)
  Nothing list
```

- funkce lze brát jako parametry, vracet z funkcí
- lambda funkce: `\vzor1 vzor2 ... vzorN -> tělo`

## Opakování: funkce vyšších řádů, lambda funkce

```
import Data.Maybe
```

```
lookBy :: (a -> Maybe b) -> [a] -> Maybe b
```

```
lookBy _ [] = Nothing
```

```
lookBy lfn (x:xs) = let mv = lfn x
                    in if isJust mv
                        then mv
                        else lookBy lfn xs
```

```
look' :: Eq a => a -> [(a, b)] -> Maybe b
```

```
look' x = lookBy (\(k, v) ->
                 if k == x then Just v else Nothing)
```

Částečná aplikace (u look')

## Pokročilejší syntax: `case`

Používání vzorů uvnitř funkce

```
lookBy :: (a -> Maybe b) -> [a] -> Maybe b
lookBy _ [] = Nothing
lookBy lfn (x:xs) = case lfn x of
    Just v -> Just v
    Nothing -> lookBy lfn xs
```

```
mapMaybe :: (a -> Maybe b) -> [a] -> [b]
mapMaybe _ [] = []
mapMaybe f (x:xs) = case f x of
    Just v -> v : mapMaybe f xs
    Nothing -> mapMaybe f xs
```

- stejně jako u vzorů funkcí se prochází odshora

# Pokročilejší syntax: strážce (guards)

Definice pomocí alternativ

```
look :: Eq a => [(a,b)] -> a -> Maybe b
look []           _ = Nothing
look ((k,v) : s) x
  | k == x       = Just v
  | otherwise    = look s x
```

- použije se definice u první podmínky, která se vyhodnotí na True
- v Prelude: otherwise = True

## Pokročilejší syntax: strážče (guards)

```
data BinTree a = BNode a (BinTree a) (BinTree a)
                | BEmpty
                deriving (Eq, Show, Read)
```

```
lookSTree :: Ord k => k -> BinTree (k, v) -> Maybe v
lookSTree _ BEmpty = Nothing
lookSTree x (BNode (k, v) l r)
    | k == x      = Just v
    | x < k       = lookSTree x l
    | otherwise   = lookSTree x r
```

```
testlst n = lookSTree n (BNode (10, "a")
                             (BNode (1, "b") BEmpty BEmpty)
                             (BNode (100, "c")
                                     (BNode (42, "?") BEmpty BEmpty)
                                     BEmpty))
```



## Pokročilejší syntax: \$

Operátor aplikace funkce

$(\$) :: (a \rightarrow b) \rightarrow a \rightarrow b$

$f \$ x = f x$

`infixr 0 $`

- aplikace funkce s nízkou prioritou a asociativitou zprava
- $f \$ g \$ x \equiv f (g x)$
- mezeru lze vnímat jako operátor aplikace s nejvyšší prioritou a asociativitou zleva
- $f g x \equiv (f g) x$

`filter (> 10) . map (^ 2) $ filter even [1..10]`

# Pokročilejší syntax: moduly

- základní modul `Prelude` načten vždy
- další načítáme pomocí `import`
- balík `Base`: <http://hackage.haskell.org/package/base>  
základní knihovna modulů
- jméno vždy velkými písmeny
- hierarchie: jméno může obsahovat tečku
- lze vytvářet nové, hlavička souboru:

```
module ModuleName (fn1, fn2, fn3) where
```

v závorkách za jménem souboru je nepovinný export funkcí a typů

## Důležité moduly v Haskell 2010/GHC

- `Data.List`: práce se seznamy
- `System.IO`: vstup a výstup
- `Data.Char`: manipulace se znaky (`isSpace`, `toLower`)
- `Data.Maybe`, `Data.Tuple`, `Data.Either` utility
- `System.Environment`: práce s příkazovou řádkou – argumenty, proměnné prostředí

## Zdroje informací

- vyhledávače Hayoo, Hoogle
- databáze balíků Hackage

## Používání v GHCi

- `:m + <modul>`

# Pokročilejší syntax: moduly, \$, IO

```
module Main (main) where

import Data.List
import Data.Char
import Data.Maybe
import System.Environment

main = do
    [filename, key] <- getArgs
    file <- readFile filename
    putStrLn $ findInFile key file

findInFile :: String -> String -> String
findInFile key = fromJust . lookup key . map splitTrim . lines
  where
    splitTrim = split . trimR . trimL
    trimL = dropWhile isSpace
    trimR = reverse . trimL . reverse
    split x = case break (== '=') x of
        (k, '=':v) -> (trimR k, trimL v)
        (k, _)      -> (k, "")
```

# Významné změny v GHC 7.10 oproti 7.8

- využití typové třídy Foldable v Prelude
  - zobecnění konceptu seznamů, přes které jde „foldovat“ na podobné datové struktury (pole, množiny, ...)
  - a odvozené funkce
  - `length :: Foldable t => t a -> Int`
  - `foldr :: Foldable t => (a->b->b) -> b -> t a -> b`
  - nahrazením `t a` za `[a]` (tj. dosazením `[]` za `t`) dostáváme známé typy na seznamech
- změny týkající se typové třídy Applicative
  - nyní přímo v Prelude
  - nově je nadtřídou typové třídy Monad (dozvíme se později, co tyto typové třídy dělají)
- důsledek: pro funkčnost kódu z 7.10 v 7.8 je často třeba doplnit importy (`Control.Applicative`), případně i importovat některé funkce z jiných modulů než Prelude (`Data.Foldable`)

Procvičte si následující příklady ze sbírky z IB015:

- vzory a větvení: **1.2.4**, alespoň 3 různými způsoby
- seznamy: **2.3.23** a **2.3.24**
- foldy: **5.1.1** a **5.1.8** alespoň část
- **6.4.8**
- **9.2.5** (co nepsat v domácích úkolech :-})