Úvod do funkcionálneho programovania

O čo ide

- štýl (prístup, paradigma) tvorby programu (ostatné paradigmy – logická, imperatívna, objektová)
- práca s funkciami a ich aplikácia na argumenty
- jednoduchosť, čitateľnosť a *elegancia* programu
- komplikované I/O, interakcia s užívateľom a okolím
- deklaratívnosť, ortogonalita, bohatý typový systém, silná typová kontrola, referenčná transparentnosť
- funkcionálne jazyky – Haskell, CaML, Lisp...
- bez príkazov, premenných a stavov
- popis, *čo* počítame, nie ako to počítame
- funkcionálny jazyk Haskell – interpret Hugs pre Windows, GHCI pre Linux

Funkcionálny program

- zložený z výrazov, celý program = výraz
- postupné vyhodnocovanie podľa lenivej redukčnej stratégie
- jednoduché výrazy:

```
5 + 2
```

```
(52 * 3) + 42
```

- výraz s lokálnou definíciou funkcie:

```
let cube y = y * y * y
   in cube 3
```

- definície na najvyššej úrovni - globálne

Príklady

- faktoriál:

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

- Fibonacciho postupnosť:

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Operátory a funkcie

- infixový a prefixový zápis (`5 + 2, (+) 5 2`)
- operátor zreťazenia (.):

```
(.) :: (b -> c) -> (a -> b) -> a -> c
(f . g) x = f (g x)
```

- čiastočná aplikácia funkcie

Typy

- každý výraz má svoj typ
- základné typy: `Bool, Char, Int, Integer, Float`...
- zložené typy: usporiadané n-tice, zoznamy
- funkcionálne typy: `Char -> Bool, (Int,Int) -> Bool, Int -> Int -> Int...`
- typová anotácia:

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

- polymorfné typy, typové premenné (`a -> a`)
- hodnoty daného typu vytvárané pomocou dátových konštruktorov

Zoznamy

- postupnosť hodnôt jedného typu ľubovoľnej dĺžky
- `[1,8,7], [['a'],['K','L','M'],[]], [(4,odd),(9,even)]`
- vzniká aplikáciou konštruktoru `(:)` na prvky a zoznamy, posledný zoznam je `[]`
- funkcie `head` a `tail`
- `(++), (!!), take, drop`
- `map, filter, zip, zipWith`
- akumulačné funkcie (`foldr`, `foldl - katamorfizmus`)

Nekonečné zoznamy

- lenivé vyhodnocovanie
- `[1..5], [2..]`
- intensionálny zápis pomocou pravidla:

```
[ 2*n | n <- [0..9] ]
```

- QuickSort:

```
qs [] = []

qs (p:s) = qs [ x | x<-s, x < p ] ++ [p] ++
qs [y|y<-s, y >= p]
```

- zoznam všetkých usporiadaných dvojíc

Binárne stromy

- `data BinTree a = Empty | Node a (BinTree a) (BinTree a)`