IB015 Neimperativní programování

Funkce na seznamech a Rekurze

Jiří Barnat
Libor Škarvada

Jak si stojíme?

Vím, že programovat znamená především přemýšlet ...

- o správném **postupu**, jak dojít k cíli,
- o **dekompozici** řešeného problému.

Vím, že k obému mi pomáha především ...

- najít vhodný způsob **uložení dat**,
- správné **otypování použitých artefaktů**.

Kudy dál?

- Musím se seznámit se základními stavebními kameny programovacího jazyka a naučit se je správně, efektivně a elegantně používat (Triforce).

Práce se seznamy v Haskellu (poprvé)

Funkce pro dekompozici seznamu

První prvek seznamu

- `head :: [a] -> a`

Seznam bez prvního prvku

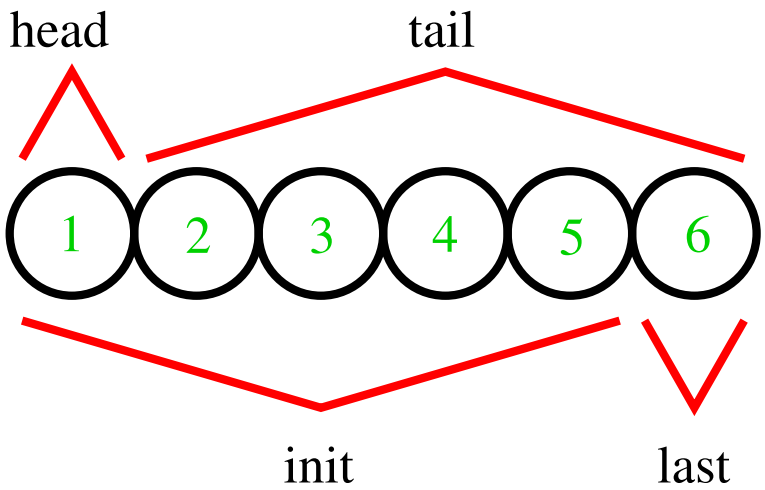
- `tail :: [a] -> [a]`

Poslední prvek seznamu

- `last :: [a] -> a`

Seznam bez posledního prvku

- `init :: [a] -> [a]`



Demonstrační cvičení

Z minula známe

- `(:)` , `(++)`

Úkol 1

- Zdvojte první prvek seznamu.
- Zduplikujte seznam a na konec přidejte první prvek.

Úkol 2

- Prohoďte první dva prvky seznamu.
- Prohoďte první a poslední prvek seznamu.

Zjištění počtu prvků v seznamu

- Vrací počet prvků v seznamu na nejvyšší úrovni, tj. nepočítá prvky v zanořených seznamech!
- `length :: [a] -> Int`

Vyzkoušejte

- `length [5,2,8]` $\rightsquigarrow$ `3`
  `length []` $\rightsquigarrow$ `0`
  `length [[]]` $\rightsquigarrow$ `1`
  `length [[3],[8,4,5,5,5,4,5]]` $\rightsquigarrow$ `2`

Prvních n prvků seznamu

- Vrátí prvních n prvků seznamu jako seznam
- `take :: Int -> [a] -> [a]`

Seznam bez prvních n prvků seznamu

- Vrátí původní seznam bez prvních n prvků
- `drop :: Int -> [a] -> [a]`

Získání n-tého prvku

- Vrátí prvek seznamu na dané pozici, první prvek je na pozici 0.
- `(!!) :: [a] -> Int -> a`

Seznam prvků v opačném pořadí

- `reverse :: [a] -> [a]`

Aritmetické a logické funkce na seznamech

- `minimum :: Ord a => [a] -> a`
- `maximum :: Ord a [a] -> a`
- `sum :: Num a => [a] -> a`
- `product :: Num a => [a] -> a`
- `or :: [Bool] -> Bool`
- `and :: [Bool] -> Bool`

Srovnejte s binárními funkcemi

- `min :: Ord a => a -> a -> a`
- `max :: Ord a => a -> a -> a`
- `(&&) :: Bool -> Bool -> Bool`
- `(||) :: Bool -> Bool -> Bool`

Demonstrační cvičení

Úkol 3

- Rozhodněte, zda je seznam palindrom.

Úkol 4

- Na neprázdných seznamech typu `Num a => [a]` definujte funkci, která rozhodne, zda je součet prvků seznamu větší, než součin.

Aplikace funkce na prvky v seznamu

- `map :: (a -> b) -> [a] -> [b]`

Vyzkoušejte

- `map not [True,False,False]` $\rightsquigarrow^*$ `[False,True,True]`
- `let f x = x + 1 in map f [4,5,6]` $\rightsquigarrow^*$ `[5,6,7]`
- `map even [3,4,5]` $\rightsquigarrow^*$ `[False,True,False]`

Výběr prvků seznamu podle dané podmínky

- `filter :: (a -> Bool) -> [a] -> [a]`

Vyzkoušejte

- `filter odd [1,2,3]` $\rightsquigarrow^*$ `[1,3]`
  `filter (not.odd) [1,2,3]` $\rightsquigarrow^*$ `[2]`

Všimněte si

- Funkce `map` i `filter` berou jako své argumenty jiné funkce. Takovým funkcím říkáme funkce vyšších řádů.

Demonstrační cvičení

Úkol 5

- Rozhodněte, zda je seznam typu `Integral a => [a]` tvořen pouze sudými čísly.

Úkol 6

- Definujte funkci, která vrátí délku zadaného seznamu, tak, abyste nepoužili funkci `length` .

Úkol 7

- Pro seznam dvojic, zaměňte první složku každé dvojice za druhou a naopak. Funkci definujte s použitím klíčového slova `where` .
- Poznámka: `where` se podobně jako `let ... in` používá pro lokální definice s tím, že klauzule `where` se vyskytuje až za hlavním výrazem.

Ponechání/odstranění podmínkou definovaného prefixu

- První prvek od začátku seznamu nevyhovující zadané podmínce definuje místo, kde končí vracený, nebo zahozený prefix.
- `takeWhile :: (a -> Bool) -> [a] -> [a]`
  `dropWhile :: (a -> Bool) -> [a] -> [a]`

Vyzkoušejte

- `takeWhile odd [1,3,5,6,7,8,9]` $\rightsquigarrow^*$ `[1,3,5]`
  `dropWhile odd [1,3,5,6,7,8,9]` $\rightsquigarrow^*$ `[6,7,8,9]`

Spojení seznamu seznamů do jednoho seznamu

- `concat :: [[a]] -> [a]`

Spojení dvou seznamů do seznamu dvojic

- `zip :: [a] -> [b] -> [(a,b)]`
- Délka výsledného seznamu je definována kratším ze dvou zadaných seznamů.

Vyzkoušejte

- `concat [[1,2],[2,3],[3,4]]` $\leadsto^*$ `[1,2,2,3,3,4]`
- `zip [1,2,3,4] "abc"` $\leadsto^*$ `[(1,'a'),(2,'b'),(3,'c')]`

Spojení seznamu pomocí funkce

- `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`

Vyzkoušejte

- `zipWith (+) [1,2,3] [1,2,3,4,5]` $\rightsquigarrow^*$ `[2,4,6]`
- `zipWith take [1,2,3] ["aaa","bbb","ccc"]`
  $\rightsquigarrow^*$ `["a","bb","ccc"]`

Trocha softwarové archeologie

Úkol

- Co počítá funkce `fn` a jakého je typu?
- ```
  fn s = and (map f (zip s (head s:s)))
    where f (a,b) = a >= b
  ```

Řešení
  ```

Trocha softwarové archeologie

Úkol

- Co počítá funkce `fn` a jakého je typu?
- ```
  fn s = and (map f (zip s (head s:s)))
    where f (a,b) = a >= b
  ```

Řešení

- `fn :: Ord a => [a] -> Bool`
- Funkce `fn` pro zadaný seznam rozhodne, zda je seznam neklesající.
  ```

Nesplnitelný úkol

Zadání

- S využitím stávajících znalostí o programovacím jazyce Haskell naprogramujte funkci `len`, která spočítá délku seznamu, a to tak, aniž byste použili funkci `length`, `sum` či jinou podobnou funkci.

Náznak řešení

- ```
  len [] = 0
  len (_:[]) = 1
  len (_:_:[]) = 2
  len (_:_:_:[]) = 3
  len (_:_:_:_:[]) = 4
  ...
  ```
  ```

Rekurze

Rekurze

Co je to rekurze

- Definice funkce, nebo datové struktury, s využitím sebe sama.

Význam v programování

- Umožňuje konečně dlouhý zápis definice funkce, která je definována pro nekonečně mnoho strukturálně odlišných parametrů.

Příklad

- Funkci `length` , která při aplikaci na seznam vrací jeho délku, je nutné definovat rekurzivně.

```
length :: [a] -> Int
length [] = 0
length (_:s) = 1 + length s
```

Příklad výpočtu rekurzivní definice

```
length :: [a] -> Int
length [] = 0
length (_:s) = 1 + length s
```

```
length [6,7,8,9]  ⇝ 1 + length [7,8,9]
                  ⇝ 1 + ( 1 + length [8,9])
                  ⇝ 1 + ( 1 + ( 1 + length [9]))
                  ⇝ 1 + ( 1 + ( 1 + ( 1 + length [])))

                  ⇝ 1 + ( 1 + ( 1 + ( 1 + 0 )))
                  ⇝ 1 + ( 1 + ( 1 + 1 ))
                  ⇝ 1 + ( 1 + 2 )
                  ⇝ 1 + 3
                  ⇝ 4
```

Rekurze na číselných funkcích

```
factorial :: Integer -> Integer

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

```
factorial 4  ⇝ 4 * factorial (4-1)
             ⇝ 4 * factorial (3)
             ⇝ 4 * ( 3 * factorial (3-1))
             ⇝ 4 * ( 3 * factorial (2))
             ⇝ 4 * ( 3 * ( 2 * factorial (2-1)))
             ⇝ 4 * ( 3 * ( 2 * factorial (1)))
             ⇝ 4 * ( 3 * ( 2 * ( 1 * factorial (0))))
             ⇝ 4 * ( 3 * ( 2 * ( 1 * 1 )))
             ⇝ 4 * ( 3 * ( 2 * 1 ))
             ⇝ 4 * ( 3 * 2 )
             ⇝ 4 * 6
             ⇝ 24
```

Práce se seznamy v Haskellu (podruhé)

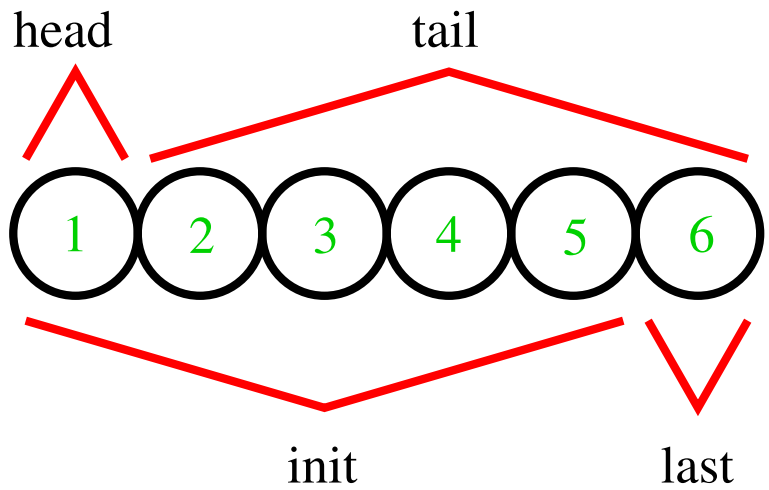
head, tail, init, last

První prvek seznamu

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

Seznam bez prvního prvku

```
tail :: [a] -> [a]
tail (_:s) = s
```



Poslední prvek seznamu

```
last :: [a] -> a
last (x:[]) = x
last (_:s) = last s
```

Seznam bez posledního prvku

```
init :: [a] -> [a]
init (_:[]) = []
init (x:_:[]) = [x]
init (x:s) = x:init s
```

null, length, (!!)

Test na prázdný seznam

```
null :: [a] -> Bool
null (_:_) = False
null [] = True
```

Délka seznamu

```
length :: [a] -> Int
length [] = 0
length (_:s) = 1 + length s
```

N-tý prvek seznamu

```
(!!) :: [a] -> Int -> a
(x:_) !! 0 = x
(_:s) !! k = s !! (k-1)
```

take, drop

Prvních n prvků seznamu

```
take :: Int -> [a] -> [a]
take _ [] = []
take n (x:s) = if (n>0) then x : take (n-1) s
                        else []
```

Seznam bez prvních n prvků;

```
drop :: Int -> [a] -> [a]
drop _ [] = []
drop n (x:s) = if (n>0) then drop (n-1) s
                        else (x:s)
```

Poznámka

- Při infixovém použití binární funkce klesá její priorita!

  x : take (n-1) s = x : (take (n-1) s)

concat, filtr, replicate

Spojení seznamů v seznamu

```
concat :: [[a]] -> [a]
concat [] = []
concat (x:s) = x ++ concat s
```

Vynechání prvků nesplňujících danou podmínku

```
filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter f (x:s) = if (f x) then x : filter f s
                          else filter f s
```

Vytvoření seznamu n-násobným kopírováním daného prvku

```
replicate :: Int -> a -> [a]
replicate 0 _ = []
replicate k x = x : replicate (k-1) x
```

takeWhile, dropWhile, map

Vynechání prvků seznamu od prvního, který nesplňuje podmínku

```
takeWhile :: (a->Bool) -> [a] -> [a]
takeWhile _ [] = []
takeWhile p (x:s) = if (p x) then x : takeWhile p s
                             else []
```

Vynechání prvků seznamu po první, který nesplňuje podmínku

```
dropWhile :: (a->Bool) -> [a] -> [a]
dropWhile _ [] = []
dropWhile p (x:s) = if (p x) then dropWhile p s
                             else x:s
```

Aplikace funkce na všechny prvky seznamu

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:s) = f x : map f s
```

zip,unzip

Spojení dvou seznamů do seznamu uspořádaných dvojic

```
zip :: [a] -> [b] -> [(a,b)]
zip [] _ = []
zip _ [] = []
zip (x:s) (y:t) = (x,y) : zip s t
```

Rozdělení seznamu dvojic na dvojici seznamů

```
unzip :: [(a,b)] -> ([a],[b])
unzip [] = ([],[])
unzip ((x,y):s) = (x : fst t, y : snd t) where t = unzip s
```

zipWith

Výpočet aplikace binární funkce na seznamy argumentů

```
zipWith :: (a->b->c)->[a]->[b]->[c]
zipWith _ _ [] = []
zipWith _ [] _ = []
zipWith f (x:s) (y:t) = f x y : zipWith f s t
```

Pozorování

```
zip = zipWith (,)
```

Demonstrace řešení vybraných příkladů (mentálně náročné)

Rekurzivní dělení dvojkou

Kolikátá mocnina dvojky

- Napište funkci, která zjistí, kolikrát se vyskytuje dvojka v prvočíselném rozkladu zadaného celého čísla.

Řešení

- Myšlenka: kolikrát mohu bezezbytku podělit dvojkou.
- ```
  kolik2 :: (Integral a, Integral b) => a -> b
  kolik2 x = y where (y,_) = f_aux (0,x)

  f_aux (cnt, rem) =
     if (rem<2) || (rem `mod` 2 /= 0)
       then (cnt,rem)
       else f_aux (cnt+1, rem `div` 2)
  ```
  ```

Zatřizování seřazených seznamů (merge)

Zatřízení sezařených seznamů

- Napište funkci, která pro dva seřazené seznamy vrátí jeden seřazený seznam, který obsahuje všechny prvky z původních dvou seznamů, tj. zatřídí dva seznamy „do sebe“.

Řešení

- ```
  merge [] x = x
  merge x [] = x
  merge (x:xs) (y:ys) = if x<y
     then x:merge xs (y:ys)
     else y:merge (x:xs) ys
  ```
  ```

Řazení seznamů (mergesort)

Řazení seznamů

- Napište funkci, která pro zadaný seznam vrátí seznam stejných prvků, ovšem seřazený podle operace `<` .

Řešení

- Seznam nejprve rozdělím na dvě části, ty setřídím rekurzivně, a pak zatřídím do sebe.

- ```
  evenodd (x:y:z) = (x:s,y:t) where (s,t) = evenodd z
  evenodd (x:[]) = ([],[x])
  evenodd _ = ([],[])
  ```

- ```
  msort [] = []
  msort [x] = [x]
  msort x = merge (msort xs) (msort ys)
     where (xs,ys) = evenodd x
  ```

Rekurze je všude

Pozorování

- Rekurzi je možné najít i na místech, kde byste ji nehledali.
- Vše co se opakuje a směřuje ke konci, lze popsat rekurzí.

Slovní úloha

- Na Jiříkovu narozeninovou párty přišlo 7 kamarádů a přineslo mimo jiné jeden narozeninový dort. Jiřík nebyl lakomý, rozdělil dort rovnoměrně mezi všechny přítomné. Jakou k tomu použil rekurzivní akci? (Jaký je typ příslušné rekurzivní funkce?)

Rekurze je všude

Pozorování

- Rekurzi je možné najít i na místech, kde byste ji nehledali.
- Vše co se opakuje a směřuje ke konci, lze popsat rekurzí.

Slovní úloha

- Na Jiříkovu narozeninovou párty přišlo 7 kamarádů a přineslo mimo jiné jeden narozeninový dort. Jiřík nebyl lakomý, rozdělil dort rovnoměrně mezi všechny přítomné. Jakou k tomu použil rekurzivní akci? (Jaký je typ příslušné rekurzivní funkce?)

Možné řešení

```
type KousekDortu = Float
dort :: [KousekDortu] -> [KousekDortu]
dort s = if (length s >= 8)
           then s
           else dort (map (/2) s ++ map (/2) s)
```

Checkpoint

Úkol 1

- Mějme konečný seznam celých čísel. Napište funkci, která vynásobí všechna čísla v seznamu zadaným parametrem, a pak ze seznamu ponechá pouze čísla dělitelná číslem 7.
- Funkci upravte tak, aby výše zmíněné číslo 7 nebylo fixně v kódu funkce, ale byl to její další parametr.

Úkol 2

- Napište funkci, která o seznamu řetězců rozhodne, zda tento seznam obsahuje alespoň 4 řetězce délky minimálně 2.
- Funkci upravte tak, aby výše zmíněné čísla 4 a 2 nebyla fixně v kódu funkce, ale byly to její další parametry.

Challenge

Challenge 1

- Napište program, který pomocí principu rekurze a s využitím odpřednášených operací na seznamech vypočítá seznam obsahující čísla od 1 do 1024. Snažte se o to, aby hloubka rekurze byla co nejmenší (logaritmická).

Challenge 2

- Jsou-li vám známy cykly s pevným počtem opakování z nějakého imperativního programovacího jazyka, popřemýšlejte o obecném postupu, jak nahradit tyto cykly voláním rekurzivní funkce.