FUNKCIONÁLIS PROGRAMOZÁS GYAKORLAT JEGYZET

Szerkesztette: Balogh Tamás

2013. május 17.

Ha hibát találsz, kérlek jelezd a *info@baloghtamas.hu* e-mail címen!

Adatok:

Név: Diviánszky Péter

E-mail: divip@aszt.inf.elte.hu

Webolal http://www.people.inf.elte.hu/divip

Szobaszám: D 2.616

Konzultáció: Csütörtök 14:00- 15:00, elötte e-mailt írni, vagy a szobájához menni

1. Gyakorlat

Haskell

Leggyakrabban használt Haskell értelmezők a *GHCi* és a *Hugs*.

Típus

`:t` paranccsal lehet lekérdezni a típusát.
`:: "típusnév"` paranccsal lehet korlátozni a típusát.

Néhány számtípus

Integer: egész szám

Int: korlátozott egész szám

Rational: racionális szám

Double: dupla pontosságú lebegőpontos szám

Pl.: `(Num a)=> a -> a-> a`
A => jobb oldalán lévő rész az argumentumok típusa, a bal oldalon lévő pedig a megszorítás.
`(/) :: Fractional a =>a-> a-> a`. Ebből következik, hogy a csak Fractional típusosztályba tartozhat.
Három különböző hatvány művelet létezik:

```
(^) :: (Integral b, Num a)=>a -> b -> a

(^^) :: (Fractional a, Integral b)=>a -> b -> a

(**) :: Floating a => a -> a -> a
```

Típuskikövetkeztetés

`1 + 2 :: Int` -> 1 és 2 is Int volt
A következö két kifejezésben az eredmény Int típusú lesz:
`"1 * 5 ::Int" "(1 :: Int)* 5"`
A különbség az, hogy míg az első kifejezésben az eredményből következetí ki, míg a másodikban 1-esből.

Zárójelezés

Operátorok precedenciája	Zárójelezésük
^, ^^, **	(...())
*, /	(()..)
+, -	(()..)
==, /=, <, <=, >, >=	-
&&	(...())
||	(...())

Az infix módon használt `div` és `mod` kötési erőssége ugyanaz, mint a * és / operátoroké.

2. Gyakorlat

Kerekítés

truncate – nulla fele kerekítés

round – legközelebbihez kerekítés

ceiling – felfele kerekítés

floor – lefele kerekítés

Enum osztály

```
toEnum :: Enum a =>Int-> a fromEnum :: Enum a =>a -> Int
```

4. Gyakorlat

Függvények

Függggvénydefiníció

```
Module First where
```

Ha ezután nem írunk semmit, akkor includeolja a *Prelude*-ot.

5. Gyakorlat

Mintaillesztés

Függvényalternatívák száma: 1, 2, 3, ...

A függvényparaméterek nem változók, hanem általában minták.

Egy minta illeszkedhet egy kifejezésre.

Az első illeszkedő alternatívát választjuk kiértékeléskor.

Listák

Alapvető listaműveletek

`!! n` operátor az n operendussal visszaadja lista n-edik elemét(0-tól indexelődik a lista).
`:` operátor hozzáfűz egy elemet a lista elejére.
`++` operátor két listát összefűz.
`head` függvény visszaadja a lista első elemét.
`tail` függvény visszaadja a lista nem első elemét.

6. Gyakorlat

Rekurzió

sum függvény

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

last függvény

```
last [y] = y
last (x:xs) = last xs
```

init függvény

```
init [y] = []
init (x:xs) = x : init xs
```

concat függvény

```
concat [] = []
concat (x:xs) = x ++ concat xs
```

minimum függvény

```
minimum [x] = x
minimum (x:xs) = min x (minimum xs)
```

++ operátor

```
(++) [] ys = ys
(++) (x:xs) ys = x: ((++) xs ys)
```

Összefésülés - merge

```
merge [] ys = ys
merge (x:xs) ys = x : merge ys xs
```

zip függvény

```
zip [x] (y:ys) = [(x,y)]
zip (x:xs) (y:ys) = (x, y) : (zip xs ys)
```

elem függvény

```
elem x [] = False
elem y (x:xs) = y == x || (elem y xs)
```

Data.List.nub függvény

```
nub [] = []
nub (x:xs) = x : nub[ z |z <- xs, x /= z ]
```

8. Gyakorlat

Esetszétválasztás

Különbség a matematikai szétválasztástól

A feltételek és az eredmények oszlopa fel van cserélve, tehát előbb írjuk a feltételt, majd = után az eredményt, valamint a { helyett | jelet írunk.

Szintaxis

Esetek száma 1, 2 ... Minden esetre: | őrfeltétel = kifejezés, ahol az őrfeltétel egy logikai típusú kifejezés, az eyenlőség jobb oldalon álló kifejezések típusa pedig megegyezik.

Szemantika

Az őrfeltételeket fentről lefelé vizsgáljuk, és az első teljesülő feltételnek megfelelő kifejezést választjuk.

Otherwise

Az `otherwise` nem kulcsszó, hanem a Prelude-ben definiált konstans, értéke igaz.

Órai példák

Nagybetű-kisbetű megcserélése

```
upperLower x
   | isUpper x = toLower x
   | otherwise = toUpper x  -- toUpper és toLower függvény nem csinál
      semmit, ha nem karaktert kap értékül.
```

digitToInt

```
digitToInt x
  | '0' <= x && x <= '9' = fromEnum x - fromEnum '0'
  | 'a' <= x && x <= 'f' = fromEnum x - fromEnum 'a' + 10
  | 'A' <= x && x <= 'F' = fromEnum x - fromEnum 'A' + 10
  | otherwise  = error ("not a digit" ++ show x)
```

Hatványozás

```
x ^ n
  | n == 0 = 1
  | odd n = x * (x ^ (n-1) )
  | otherwise = sqr ( x ^ div n 2)
```

11. Gyakorlat

Magasabb rendű függvények

Pascal háromszög

Pascal triangle

```
pascalTriangle = iterate f [1] where
   f xs = zipWith (+) xs (0 : xs) ++ [1]
```

Négyzetgyök számolás Newton-módszerrel statikusan

sqrt

```
sqrt a = iterate (\x -> (x+a/x)/2) a !! 10
```

12. Gyakorlat

Függvénykompozíció

Függvénykompozíció

```
(.) f g x = f (g x)
```

1, 11, 111.. lista előállítása

```
(.) f g x = f (g x)
```

numbersMadeofOnes

```
numbersMadeofOnes = iterate((+1) .  (*10)) 1
```

numbersMadeofThrees

```
numbersMadeOfThrees = map (3*) numbersMadeofOnes
```

firstLetters

```
firstLetters = unwords . map(take 1) . words
```

monogram

```
monogram = unwords . map (( ++ ". ") . take 1) . words
```

14. Gyakorlat

type, newtype, data

Sakktábla mező előállítása

```
s (a, b)
   |elem a ['A'..'H'] && elem b [1..8] = S(a, b)
   |otherwise = error "hiba"
```

Tükrözés pontra

```
mirrorP (P(a,b)) (P(c,d))
  | a <= c && b <= d= P(a-c, b-d)
  | a > c && b > d =P(a+(a-c), b+(b-d))
```

Forrás

- ELTE IK programtervezői informatikus szak 2013 tavaszi féléves Funkcionális programozás gyakorlat alapján írt órai jegyzetem.
- `http://pnyf.inf.elte.hu/fp/Index.xml`