Funkcionális programozás
3. előadás

MÁRTON Gyöngyvér

Sapientia Egyetem, Matematika-Informatika Tanszék
Marosvásárhely, Románia
mgyongyi@ms.sapientia.ro

2025, tavaszi félév

Miről volt szó?

Haskell:

- Az első lépések
- GHC parancsok
- Alaptípusok
- feladatok: faktoriális számítás
- típusosztályok
- típusdefiníciók
- megjegyzések használata
- könyvtármodul importálása

Miről lesz szó?

- feltételek megadása
- rekurzió, margószabály, mintaillesztés
- halmazkifejezések,
- magasabb rendű függvények,
- lambda kifejezések
- magasabb rendű függvények részleges paraméterezése

Feltételek megadása (definition by cases)

Feltételek megadása: a függvény értékét az első, igaz feltételhez tartozó kifejezés adja, amely kifejezést az aktuális paraméter értéke alapján értékeljük ki.

1. feladat

Határozzuk meg egy szám előjelét.

```
elojel :: Int -> Int
elojel x
   | x < 0 = -1
   | x > 0 = 1
   | x == 0 = 0
```

```
elojel1 :: Int -> String
elojel1 0 = "nulla"
elojel1 x
   | x < 0 = "negatív"
   | x > 0 = "pozitív"
```

a függvény kiértékelése:

```
> elojel (-10)
```

Rekurzió (recursion)

A rekurzió a funkcionális nyelvek alapvezérlési szerkezete, a függvények hivatkozhatnak önmagukra és kölcsönösen egymásra.

2. feladat

Határozzuk meg két egész szám legnagyobb közös osztóját.

Kivonásos módszer

```
lnko :: Int -> Int -> Int
lnko a b
       | a > b = lnko (a-b) b
       | a < b = lnko a (b-a)
       | otherwise = a
```

Eukleidész módszere

```
euklid :: Integral a => a -> a -> a
euklid a b
   | b == 0 = a
   | otherwise = euklid b (mod a b)
```

a függvény kiértékelése:

```
> lnko 24 204
```

Margószabály (layout rule)

Az összetartozó kifejezéseket a baloldali margó alapján lehet megállapítani.

3. feladat

Határozzuk meg egy másodfokú egyenlet valós gyökeit.

```
masodE :: (Fractional a, Floating a, Ord a) => a -> a -> a -> (a, a)
masodE a b c
   | delta < 0 = error "Komplex gyokok"
   | otherwise  = (x1, x2)
   where
       x1 = (-b + sqrt delta) / n
       x2 = (-b - sqrt delta) / n
       delta = b * b - 4 * a * c
       n = 2 * a

delta a = 3 * a
```

a függvény kiértékelése:

```
> masodE 2 3 1
```

Melyik `delta` kifejezés értékelődik ki?

```
> delta 2
```

Mintaillesztés (pattern matching)

Az argumentumok mintaillesztése: a függvényértékét az a függvénytörzs határozza meg, amelyre a formális paraméter egy megadott minta alapján illeszkedik. A mintaillesztést és a feltételek megadását lehet együttesen is alkalmazni.

4. feladat

Határozzuk meg egy szám számjegyeinek összegét, szorzatát.

```
szOsszeg :: Int -> Int
szOsszeg 0 = 0
szOsszeg x = mod x 10 + szOsszeg (div x 10)

szSzorzat :: Int -> Int
szSzorzat 0 = 0
szSzorzat x
   | x < 10 = x
   | otherwise = mod x 10 * szSzorzat (div x 10)
```

Mintaillesztés

5. feladat

Határozzuk meg egy 3 elemű lista elemeinek összegét.

```
osszeg :: [Int] -> Int
osszeg [] = 0
osszeg [x] = x
osszeg [x,y] = x + y
osszeg [x,y,z] = x + y + z
```

a függvény kiértékelése:

```
> osszeg [10, 4, 2]
16
```

Mi történik a 4, 5 stb elemű listák esetében? Futási hiba adódik, ezek az esetek nincsenek letárgyalva:

```
> osszeg [10, 4, 2, 8]
*** Exception: ...: Non-exhaustive patterns in function osszeg
```

Halmazkifejezések (list comprehension)

Iteratív adatszerkezetek (listák, halmazok, sorozatok) elemeinek megadására alkalmazott jelölésrendszer.

6. feladat

Határozzuk meg a paraméterként megadott szám osztóit.

```
osztok :: Int -> [ Int ]
osztok n = [ i | i <- [1..n], mod n i == 0 ]
```

a függvény kiértékelése:

```
> osztok 60
[1,2,3,4,5,6,10,12,15,20,30,60]

> length (osztok 60)
12
```

Magasabb rendű függvények (high order function)

- argumentumuk lehet függvény, és visszatérítési értékük is lehet függvény,
- könyvtárfüggvények: map, filter, foldr, foldl, stb.
- map: két argumentuma van, az első egy függvény, amelyet alkalmaz a listaként megadott második argumentumára.

7. feladat

Határozzuk meg a paraméterként megadott számok négyzetgyökét.

```
> map sqrt [ 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 ]

[1.7320508075688772,2.0,2.23606797749979,2.449489742783178,
2.6457513110645907,2.8284271247461903,3.0]
```

Magasabb rendű függvények

8. feladat

Határozzuk meg a paraméterként megadott számok közül a páros számokat.

```
parosLista :: (Integral a) => [a] -> [a]
parosLista ls = filter even ls

> paroslista [1..20]

[2,4,6,8,10,12,14,16,18,20]
```

Az `even` könyvtárfüggvény, amelynek `True` a visszatérési értéke, ha a bemenet páros szám, ellenkező esetben `False`.

```
> even 10
True
```

Magasabb rendű függvények

- argumentumuk lehet függvény, és visszatérítési értékük is lehet függvény,
- könyvtárfüggvények: map, filter, foldr, foldl, stb.

9. feladat

Növeljük a paraméterként megadott szám értékét kettővel.

```
duplaz :: (a -> a) -> a -> a
duplaz f x = f (f x)

myInc :: (Num a) => a -> a
myInc x = x + 1

> duplaz myInc 10
12

> duplaz (* 10) 10
1000
```

A `duplaz` első paramétere egy függvény, amit kétszer alkalmaz a második paraméterére, a `my_inc` a megadott paraméter értékét 1-el növeli.

Lambda kifejezések, névtelen függvények

- a függvények egy alternatív definiálási módja, akkor használjuk, ha nem akarunk nevet választani egy adott segédfüggvénynek,
- nem tartalmazhatnak őrfeltételeket, mintaillesztéseket,
- vigyázni kell hogy a függvénydefiníció minden lehetséges helyzetre kiértékelődjön.

10. feladat

Növeljük a paraméterként megadott szám értékét 1-el.

```
myIncL :: (Num a) => a -> a
myIncL = \x -> x + 1
```

11. feladat

Növeljük egy listában megadott számok értékét 1-el.

```
listInc1 :: (Num a) => [a] -> [a]
listInc1 ls = map (\x -> x + 1) ls

listInc2 :: (Num a) => [a] -> [a]
listInc2 ls = [(\x -> x + 1) k | k <- ls]

> listInc1 [1..10]
[2,3,4,5,6,7,8,9,10,11]
```

Magasabb rendű függvények, részleges paraméterezés

- partial parameterization, curry-zés, Haskell Curry matematikus után,
- a függvényhívás megengedett kevesebb paraméterrel is.

12. feladat

Írjunk egy Haskell-függvényt, amely az `x` *és* `k` *bemenetekre, ahol* `k` *egész szám, meghatározza az* x^0, x^1, ..., x^k *értékeket.*

```
fugv1 :: (Integral a, Num b) => b -> a -> [b]
fugv1 x k = map (x ^) [0..k]

> fugv1 7 6
[1,7,49,343,2401,16807,117649]
```

A `map` függvény első paraméterét a hatványozó operátort, infix formában, részlegesen paraméterezve adtuk meg.

Magasabb rendű függvények, részleges paraméterezés

13. feladat

Írjunk egy Haskell-függvényt, amely az `x` *és* `k` *bemenetekre meghatározza az* 0^k, 1^k, ..., x^k *értékeket.*

A ^ operátort infix formában hívjuk úgy, hogy a ^ operátor első argumentuma rendre a `[0..x]` lista elemei legyen.

```
fugv2 :: (Integral a, Num b, Enum b) => b -> a -> [b]
fugv2 x k = map (^ k) [0..x]

> fugv2 7 6
[0,1,64,729,4096,15625,46656,117649]
```

Módosítjuk a zárójelezést, a ^ operátor prefix formában kerül meghívásra, más lesz az eredmény: k^0, k^1, ..., k^x

```
fugv2_ :: (Integral a, Num b) => a -> b -> [b]
fugv2_ x k = map ((^) k) [0..x]

> fugv2_ 7 6
[1,6,36,216,1296,7776,46656,279936]
```

Magasabb rendű függvények, részleges paraméterezés

A beépített operátor helyett megírjuk a saját hatványozó függvényünket, a `map` függvénynek ezt adjuk meg paraméternek. A `myPow1` függvény első paramétere az alap, második a hatványkitevő lesz.

```
myPow1 :: (Integral a) => a -> a -> a
myPow1 x n
    | n < 0 = error "Negativ kitevo"
    | n == 0 = 1
    | even n = temp * temp
    | otherwise = x * temp * temp
      where
      temp = myPow1 x (div n 2)
```

Magasabb rendű függvények, részleges paraméterezés

A `fugvA`-ban prefix formában használjuk `myPow1` függvényt, azért hogy az x^0, x^1, ..., x^k értékeket határozza meg:

```
fugvA :: Integral a => a -> a -> [a]
fugvA x k = map (myPow1 x) [0..k]

> fugvA 7 6
[1,7,49,343,2401,16807,117649]
```

A `fugvB`-ben infix formában, azért hogy a 0^k, 1^k, ..., x^k értékeket határozza meg:

```
fugvB :: Integral a => a -> a -> [a]
fugvB x k = map (`myPow1` k) [0..x]

> fugvB 7 6
[0,1,64,729,4096,15625,46656,117649]
```

Magasabb rendű függvények, részleges paraméterezés

A `flip` függvény:

```
> myPow1 2 10
1024
> flip myPow1 2 10
100
```

A `fugvC`-ben alkalmazásra kerül a beépített `flip` függvény, amely segítségével a paraméterek sorrendjét lehet megváltoztatni, így most is a 0^k, 1^k, ..., x^k értékeket határozza meg a kiértékelés.

```
fugvC :: Integral a => a -> a -> [a]
fugvC x k = map (flip myPow1 k) [0..x]

> fugvC 7 6
[0,1,64,729,4096,15625,46656,117649]
```

Magasabb rendű függvények, részleges paraméterezés

14. feladat

Válasszuk ki egy adott listából az x-el osztható számokat, használjuk a `filter` *könyvtárfüggvényt:*

```
oszthato :: (Integral a) => a -> a -> Bool
oszthato x y
   | mod y x == 0 = True
   | otherwise = False

oszthato_ :: (Integral a) => a -> a -> Bool
oszthato_ x y = mod y x == 0

fugv :: (Integral a) => a -> [a] -> [a]
fugv x ls = filter (oszthato x) ls

fugv1 x = filter (flip (\x y -> mod x y == 0) x)

> fugv1 7 [1..100]
[7,14,21,28,35,42,49,56,63,70,77,84,91,98]
```