Funkcionális programozás
3. előadás

MÁRTON Gyöngyvér

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

2024, tavaszi félév

Miről volt szó?

- típusosztályok
- típusdefiníciók
- megjegyzések használata
- könyvtármodul importálása
- feltételek megadása
- rekurzió, margószabály, mintaillesztés
- halmazkifejezések,
- magasabb rendű függvények,
- feladatok:
  - területszámítás
  - abszolút érték
  - aritmetikai műveletek
  - tuple elemek megegyeznek-e?
  - másodfokú egyenlet gyökei
  - legnagyobb közös osztó
  - számjegyek összege, szorzata
  - szám osztóinak listája

Miről lesz szó?

- lambda kifejezések
- magasabb rendű függvények részleges paraméterezése
- a `list`, `tuple` típusok, operátorok, függvények listákon
- kifejezések:
  - `if ... then ... else`
  - `let ... in`
- feladatok:
  - gyorshatványozás
  - két pont közötti távolság
  - tetszőleges számrendszerben a számjegyek száma
  - gyorshatványozás
  - másodfokú egyenlet gyökei

Magasabb rendű függvények

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

1. 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.

2. feladat

Növeljük a paraméterként megadott szám értékét 1-el.

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

3. 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.

4. feladat

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

5. 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

6. 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

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

A lista típus

- ugyanolyan típusú elemek sorozata, ahol az elemek száma változó. Jelölésére a szögletes zárójelt használjuk: `[]`, sorszámozásuk nullától kezdődik.
  - `[]` - egy üres listát mintáz,
  - `[x]` - egy egyelemű listát mintáz,
  - `[x, y]` - egy kételemű listát mintáz,
  - `(k : ve)` - egy olyan listát mintáz, melynek első eleme `k`, `ve` pedig a lista vége, ahol `k` elem, `ve` lista típusú,
- Függvények listákon, lista elemeinek összege (`sum`), lista hossza (`length`), lista elemeinek megfordítása (`reverse`), stb:

```
> sum [3, 2, 10, 7, 5]
27
> length [3, 2, 10, 7, 5]
5
> reverse [3, 2, 10, 7, 5]
[5,7,10,2,3]
```

Operátorok listákon

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

- hozzáad egy új elemet a listához, amelyet a lista elejére tesz, a típusdefiníció általános lista feldolgozását teszi lehetővé,
- hozzárendeli a lista első elemét egy azonosítóhoz, a lista többi elemét, pedig egy másik nevű azonosítóhoz rendeli hozzá

```
> ls1 = [1, 2, 3, 4]
> ls1_ = 0 : ls1
> print ls1_
[0, 1, 2, 3, 4]
```

```
> k : ve = "Hello Vilag"
> print k
'H'
> print ve
"ello Vilag"
```

```
> ls2 = "apientia"
> ls2_ = 'S': ls2
> print ls2_
"Sapientia"
```

Operátorok listákon

```
> ls3 = [[1,2,3,4], [1,2,3], [1,2]]
> ls3_ = [1..5] : ls3
> print ls3_
[[1,2,3,4,5], [1,2,3,4], [1,2,3], [1,2]]

> ls4 = [0,5..40]
> print ls4
[0, 5, 10, 15, 20, 25, 30, 35, 40]

> ls5 = [-3, -6.. -20]
> print ls5
[-3, -6, -9, -12, -15, -18]
```

Operátorok listákon

```
> ls6 x = [2^x, length (show (2 ^ x))]
> print ls6 10
[1024,4]

> ls7 x = 3^x : length (show (3^x)) : (ls6 x)
> print ls7 10
[59049, 5, 1024, 4]

> ls8 = ['a'..'z']

> ls9 = ['A' ..'Z'] ++ ls8
> print ls9
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

> ls9 !! 2
'C'
```

Függvények listákon

`head :: [a] -> a`
visszatéríti egy lista első elemét, nem alkalmazható üres listára, a típusdefiníció általános lista feldolgozását teszi lehetővé.

```
myHead :: [a] -> a
myHead [] = error "ures lista"
myHead (k : ve) = k

> myHead [1..10]
1

> myHead ["alma", "korte", "barack", "szilva"]
"alma"

> head [1..10] -- a konyvtarfuggveny
1
```

Függvények listákon

`tail :: [a] -> [a]`
egy listát térít vissza, amelyben nem szerepel az eredeti lista első eleme, nem alkalmazható üres listákra.

```
myTail :: [a] -> [a]
myTail [] = error "ures lista"
myTail (k : ve) = ve

> myTail [1..10]
[2,3,4,5,6,7,8,9,10]

> myTail ["alma", "korte", "barack", "szilva"]
["korte", "barack", "szilva"]

> tail [1..10] -- a konyvtarfuggveny
[2,3,4,5,6,7,8,9,10]
```

Függvények listákon

```
> reverse "Sapientia"
"aitneipaS"
> maximum "erdelyi magyar  tudomanyegyetem"
'y'
> head "Keleti-Karpatok"
'K'
> tail ["Kelemen", "Gyergyoi", "Hargita", "Csalho"]
["Gyergyoi", "Hargita", "Csalho"]
> null []
True
> init ["Kelemen", "Gyergyoi", "Hargita", "Csalho"]
["Kelemen","Gyergyoi","Hargita"]
> last "Nagy-Hagymas"
's'
> last ["Kelemen", "Gyergyoi", "Hargita", "Csalho"]
"Csalho"
```

A rendezett n-es (tuple) típus

- különböző típusú elemek halmaza, ahol, az elemek száma rögzített. Jelölésére a kerek zárójelt használjuk: ().

```
> myTuple = ("Marika", 3, 8.75)
> (nev, evf, jegy) = myTuple
> print nev
> "Marika"
> print evf
> 3
> print jegy
> 8.75
```

- egy 2 elemű tuple típuson alkalmazhatóak az `fst` és `snd` könyvtárfüggvények:

```
> myTuple1 = ("Marika", 8.75)
> fst myTuple1
> "Marika"
> snd myTuple1
> 8.75
```

A tuple típus

Három, vagy többelemű tuple-ok esetében nem működik az `fst`, az `snd`, de könnyedén megírhatóak:

```
> myFst (t1, t2, t3) = t1
> myFst ("Mari", 1990, 8.50)
"Mari"

> mySnd (t1, t2, t3) = t2
> mySnd ("Mari", 1990, 8.50)
1990

> myThd (t1, t2, t3) = t3
> myThd ("Mari", 1990, 8.50)
8.5
```

A tuple típus

7. feladat

A következő `tupleMax` meghatározza a paraméterként megadott háromelemű tuple típusú adatok közül azt, amelyiknek a harmadik eleme a nagyobb.

```
tupleMax :: (String, Int, Double) -> (String, Int, Double)
              -> (String, Int, Double)
tupleMax t1 t2 = if m == x3 then t1 else t2
    where
    (x1, x2, x3) = t1
    (y1, y2, y3) = t2
    m = max x3 y3

> tupleMax ("Mari", 1990, 8.50) ("Feri", 1991, 9.25)
("Feri",1991,9.25)
```

A `let...in` kifejezés

Lokális kifejezések, azonosítók definiálására használhatjuk.

8. feladat

A következő `tupleMin` meghatározza a paraméterként megadott háromelemű tuple típusú adatok közül azt, amelyiknek a második eleme a kisebb.

```
tupleMin :: (String, Int, Double) -> (String, Int, Double)
              -> (String, Int, Double)
tupleMin t1 t2 =
    let
        m = min x y
        x = mySnd t1
        y = mySnd t2
    in
        if m == x then t1 else t2

> tupleMin ("Mari", 1990, 8.50) ("Feri", 1991, 9.25)
("Mari",1990,8.5)
```

A let ... in kifejezés

9. feladat

Határozzuk meg egy másodfokú egyenlet valós gyökeit.

```
masodE :: (Floating a, Ord a) => a -> a -> a -> (a, a)
masodE a b c =
    let
        x1 = (-b + sqrt delta) / n
        x2 = (-b - sqrt delta) / n
        delta = b * b - 4 * a * c
        n = 2 * a
    in
        if delta < 0 then  error "Komplex gyokok"
        else (x1, x2)

> masodE 1 3 2
(-1.0,-2.0)
```

A type kulcsszó

A type kulcsszóval új típusnevek adhatók, azaz típusszinonimákat tudunk létrehozni.

10. feladat

Definiáljunk egy Pont típusú értéket és írjunk három függvényt: definiáljuk a kezdőpontot, mozgassuk el a pontot, határozzuk meg két pont között a távolságot.

```
type Szin = String
type Pont = (Double, Double, Szin)

kezdoP :: Szin -> Pont
kezdoP szin = (0, 0, szin)

mozgat :: Pont -> Double -> Double -> Pont
mozgat (x, y, szin) xTav yTav = (x + xTav, y + yTav, szin)

tavolsag :: Pont -> Pont -> Double
tavolsag (x1, y1, szin1) (x2, y2, szin2) = sqrt (dx * dx + dy * dy)
   where
   dx = x2 - x1
   dy = y2 - y1

> p1 = kezdoP "fekete"
> p2 = mozgat p1 10 15
> tavolsag p1 p2
18.027756377319946
```

A tuple típus

11. feladat

Definiáljunk egy Haskell függvényt, amely meghatározza egy szám `b` számrendszerbeli alakjában a `d`-vel egyenlő számjegyek számát, majd alkalmazzuk a `map` függvényt, illetve halmazkifejezést is írjunk.

```
bSzamjegy :: (Integral a) => (a, a, a) -> a
bSzamjegy (nr, b, d)
    | nr < b && nr == d = 1
    | nr < b && nr /= d = 0
    | m == d = 1 + tmp
    | m /= d = tmp
        where
        m = mod nr b
        tmp = bSzamjegy (div nr b, b, d)

bSzamjegyMap :: (Integral a) => [(a, a, a)] -> [a]
bSzamjegyMap = map bSzamjegy

> bSzamjegyMap [(1024, 2, 1), (1024, 2, 0), (1023, 10, 3), (767676, 10, 6)]

bSzamjegyList :: (Integral a) => [(a, a, a)] -> [a]
bSzamjegyList ls = [bSzamjegy i | i <- ls]

> bSzamjegyList [(1024, 2, 1), (1024, 2, 0), (1023, 10, 3)]
```

Az `if ... then ... else` kifejezés

12. feladat

Az `if ... then ... else` *kifejezést alkalmazva határozzuk meg* `n!`*-t.*

```
faktorialis :: Integer -> Integer
faktorialis n = if n == 0 then 1 else n * faktorialis4 (n-1)
```

Az if a Haskell-ben egy kifejezés, az else ág kötelező.

13. feladat

Határozzuk meg x^n*-t.*

```
myPow2 :: (Integral a) => a -> a -> a
myPow2 x n =
   if n < 0 then error "negativ kitevo"
   else
       if n == 0 then 1
       else
           if mod n 2 == 0 then myPow2 (x * x) (div n 2)
           else  x * myPow2 (x * x) (div n 2)
```