Funkcionális programozás
3. előadás

MÁRTON Gyöngyvér

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

2023, tavaszi félév

Miről volt szó?

- típusosztályok
- típusdefiníciók
- megjegyzések használata
- könyvtármodul importálása
- feltételek megadása
- rekurzió, margószabály, mintaillesztés
- halmazkifejezések, lambda kifejezések
- magasabb rendű függvények, részleges paraméterezés
- a `list` típus, operátorok, függvények listákon
- feladatok:
  - területszámítás
  - abszolút érték
  - aritmetikai műveletek
  - tuple elemek megegyeznek-e?
  - másodfokú egyenlet gyökei
  - legnagyobb közös osztó
  - számjegyek összege, szorzata
  - szám osztóinak listája
  - gyorshatványozás

Miről lesz szó?

- a `list, tuple` típusok, operátorok, függvények listákon
- kifejezések:
  - `if ... then ... else`
  - `case ... of`
  - `let ... in`
- függvénykompozíció
- függvénykiértékelés a $ szimbólummal
- a $ és a . szimbólumok
- feladatok:
  - két pont közötti távolság
  - tetszőleges számrendszerben a számjegyek száma
  - gyorshatványozás
  - hexa szimbólumok
  - másodfokú egyenlet gyökei
  - négyzetszámok, összetett számok, prímszámok tesztelése

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,

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

- hozzáad egy új elemet a lista elejére tesz,
- hozzárendeli a lista első elemét egy azonosítóhoz, a lista többi elemét, pedig egy másikhoz,

```
> ls1 = [31, 12, 53, 74]
> ls1_ = 0 : ls1
> print ls1_
[0, 31, 12, 53, 74]
```

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

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

1. 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.

2. 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

3. 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

4. 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 `type` kulcsszóval új típusnevek adhatók, azaz típusszinonimákat tudunk létrehozni.

A tuple típus

5. 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

6. 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.

7. 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)
```

A case ... of kifejezés

8. feladat

Határozzuk meg egy 16-os számrendszerben megadott számsorozatnak azon alakját, amelyben a számértékeket a 16-os számrendszerben használt szimbólumokkal helyettesítjük.

```
import Data.Char
hexaSz :: Int -> Char
hexaSz c
    | c >= 0 && c < 16 =
        case c of
            10 -> 'A'
            11 -> 'B'
            12 -> 'C'
            13 -> 'D'
            14 -> 'E'
            15 -> 'F'
            _ -> chr (c + 48)
    | otherwise = error "rossz bemenet"
```

```
> map hexaSz [12, 4, 5, 10, 15]
"C45AF"

> [hexaSz x | x <- [10, 12, 3, 9, 1, 11]]
"AC391B"

> map hexaSz [10, 16, 6]
"A*** Exception: rossz bemenet...
```

Feladat

9. feladat

Írjunk egy Haskell-függvényt, amely meghatározza egy 16-os számrendszerben megadott számsorozatnak azon alakját, amelyben a számértékeket a 16-os számrendszerben használt szimbólumokkal helyettesítjük. Használjuk a korábban megírt `hexaSz` függvényt.

```
hexaLs1 :: [Int] -> [Char]
hexaLs1 [] = ""
hexaLs1 (k : ve) = hexaSz k : hexaLs1 ve

hexaLs2 :: [Int] -> [Char]
hexaLs2 ls = map hexaSz ls

hexaLs3 :: [Int] -> [Char]
hexaLs3 ls = [hexaSz k | k <- ls]

> hexaLs2 [12, 4, 5, 15, 7, 0, 11, 4]
"C45F70B4"
```

Függvénykompozíció

A matematikából ismert művelet megfelelője.

10. feladat

Határozzuk meg a paraméterként megadott számok közül a páratlan számokat.

```
paratlanLista :: (Integral a) => [a] -> [a]
paratlanLista ls = filter (not . even) ls

> paratlanLista [1..20]
[1,3,5,7,9,11,13,15,17,19]
```

Nem kell explicit módon megadni a paramétert:

```
paratlanLista1 :: (Integral a) => [a] -> [a]
paratlanLista1 = filter (not . even)

> paratlanLista1 [1..20]
[1,3,5,7,9,11,13,15,17,19]
```

Függvénykompozíció

A korábbi előadásban megadott `duplaz` függvény is megadható függvénykompozíciót alkalmazva:

```
duplaz :: (a -> a) -> a -> a
duplaz fg = fg . fg

> duplaz (+1) 10
```

11. feladat

Határozzuk meg a paraméterként megadott természetes számokat tartalmazó listából azokat a számokat, amelyek nem négyzetszámok.

```
nemNegyzet :: (Integral a) => [a] -> [a]
nemNegyzet = filter (not . negyzetV)

negyzetV :: Integral a => a -> Bool
negyzetV x = temp * temp == x
    where
    temp = truncate (sqrt (fromIntegral x))

> nemNegyzet [12, 121, 34, 49, 625, 133]
[12,34,133]
```

Függvénykompozíció

12. feladat

Vágjuk le a paraméterként megadott lista, karakterlánc első és utolsó karakterét.

Alkalmazzuk az `init, tail` könyvtárfüggvényeket.

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

levag :: [a] -> [a]
levag = init . tail

> levag "gezakekazeg"
"ezakekaze"
```

Függvénykompozíció

13. feladat

Írjunk egy Hakell függvényt, amely a paraméterként megadott 1-nél nagyobb, természetes számokat tartalmazó listából, kiválogatja az összetett számokat.

```
osszetett :: (Integral a) => [a] -> [a]
osszetett ls = filter (not . primT 3) ls

osszetett1 :: (Integral a) => [a] -> [a]
osszetett1 ls = [i | i <- ls, (not . primT 3) i]

import Data.List
osszetett2 :: (Integral a) => [a] -> [a]
osszetett2 ls = ls \\ [i | i <- ls, primT 3 i]

> osszetett [24, 97, 5, 1, 74, 41, 61, 19, 100]
[24,1,74,100]
```

Függvénykompozíció

A primtesztelést végző függvény:

```
primT :: (Integral a) => a -> a -> Bool
primT k nr
    | nr < 0 = error "hibas bemenet"
    | nr == 1 = False
    | nr == 2 = True
    | mod nr 2 == 0 = False
    | nr < k * k = True
    | mod nr k == 0 = False
    | otherwise = primT (k + 2) nr

> primT 3 101
True
```

A $ operátor

- a kifejezések kiértékelési sorrendjét változtatja meg,
- fölöslegessé válik a zárójelezés,
- jobbról asszociatív: előbb a jobb oldalon levő kifejezés értékelődik ki,
- az operátorok között legkisebb a prioritása.

Meghatározza $\sqrt{2}$, majd hozzáadja a 3-t, majd az 5-t:

```
> sqrt 2 + 3 + 5
9.414213562373096
```

Összeadja a számokat és azután határozza meg $\sqrt{10}$ értékét:

```
> sqrt (2 + 3 + 5)
3.1622776601683795
```

Összeadja a számokat és azután határozza meg $\sqrt{10}$ értékét:

```
> sqrt $ 2 + 3 + 5
3.1622776601683795
```

Függvénykiértékelés a $ szimbólummal

- Előbb alkalmazza az `abs` függvényt:

```
> sqrt $ abs (-16)
4.0
```

- Összeadja a számokat, alkalmazza az `abs` függvényt, majd meghatározza a négyzetgyököt:

```
> sqrt $ abs $ (-16) + 9
2.6457513110645907
```

- Alkalmazza az `abs` függvényt, hozzáadja a 9-et, majd meghatározza a négyzetgyököt:

```
> sqrt $ abs (-16) + 9
5.0
```

A $ és a . szimbólumok

A . szimbólummal elsősorban az függvényhívások össze-láncolását (kompozicióját) lehet megvalósítani, a lényeg nem a kevesebb zárójelhasználat, bár az eredmény az, hogy kevesebb zárójelt kell használni.

- Előbb alkalmazza az `abs` függvényt:

```
> (sqrt . abs) (-16)
4.0
```

- Ugyanazt érem el, mint az előbbi, de kevesebb zárójellel:

```
> sqrt . abs $ -16
4.0
```

A $ és a . szimbólumok

- előbb alkalmazza a `head` függvényt, majd a `toLower` függvényt és utána a `:` konstruktort, azaz épít egy új listát aminek az első elemét kisbetűre cseréli, a többi marad változatlan

```
import Data.Char
fugvKB :: [Char] -> [Char]
fugvKB ls = (toLower . head $ ls) : tail ls

> fugvKB "SAPIENTIA"
"sAPIENTIA"
```

- a karakterlánc első betűjét nagybetűre cseréli, a többi marad változatlan

```
fugvNB :: [Char] -> [Char]
fugvNB ls = (toUpper . head) ls : tail ls

> map fugvNB ["sapientia", "emte", "mvh"]
["Sapientia","Emte","Mvh"]
```

A $ és a . szimbólumok

A `words` a paraméterként megadott stringet szavakra bontja:

```
> words "sapientia hungarian university marosvasarhely"
["sapientia","hungarian","university","marosvasarhely"]

> ls = "sapientia hungarian university marosvasarhely"
> map fugvNB $ words ls
["Sapientia","Hungarian","University","Marosvasarhely"]
```

Másképp:

```
> map (\x -> (toUpper . head) x : tail x) (words ls)

> (map (\x -> (toUpper . head) x : tail x) . words) ls

> map (\x -> (toUpper . head) x : tail x) $ words ls
```

A $ és a . szimbólumok

Az `unwords` a fordított műveletet végzi úgy, hogy az összefűzött szavak közé szóközöket tesz:

```
> unwords ["paring","fogaras","kiralyko"]
"paring fogaras kiralyko"
```

A szavak kezdőbetűit átalakítja nagybetűkké, 2 módszerrel:

```
fugvM1 :: String -> String
fugvM1 ls =  unwords $ map aux $ words ls
    where
    aux = \ x -> (toUpper . head) x : tail x

fugvM2 :: String -> String
fugvM2 = unwords . map aux . words
    where
    aux x = (toUpper . head) x : tail x

> fugvM1 "retyezat kudzsiri bucsecs jezer"
"Retyezat Kudzsiri Bucsecs Jezer"

> fugvM2 "szebeni csernai vulkan"
"Szebeni Csernai Vulkan"
```