Funkcionális programozás
12. előadás

MÁRTON Gyöngyvér

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

2023, tavaszi félév

Miről volt szó?

- kombinatorikai feladatok
  - általánosan:
    - a lexicografikus sorrend,
    - az `n` elem `m`-ed rendű kombinációi feladat,
    - az `n` királynő feladat
  - részhalmazok előállítása
  - Pascal-háromszög
- `ByteString`-ek

Miről lesz szó?

- JSON formátumú adatok

JSON formátumú adatok

- JSON formátumú adatokat, állományokat először JavaScript-el dolgoztak fel,
- az adatok emberek által is olvasható formátumban vannak, a szerkezet kötött, több típusú adat tárolása biztosított: számok, logikai értékek, karakterláncok, egydimenziós tömbök, objektumok,
- a Haskell a `Data.Aeson`, illetve `Data.Aeson.Encode.Pretty` könyvtármodulok importálásával nyújt lehetőséget JSON típusú adatok kezelésére
- ezek installálása, Windows alatt a `PowerShell`-en, Visual Studio Code, vagy valamilyen terminálon keresztül a következőképpen történik:

```
> cabal v1-install aeson
> cabal v1-install aeson-pretty
```

JSON formátumú adatok

- A `Data.Aeson` könyvtármodulban definiált `ToJSON` és `FromJSON` típusosztályok segítségével létrehozhatunk különböző típusú adatokból JSON típusú adatokat, illetve JSON típusú adatokat alakíthatunk át tetszőleges típusú adattá.
- JSON típusú adatfeldolgozásnál a `Generic` típusosztály biztosítja az adatok automatikus konverzióját; használatához a `GHC.Generics` importálása szükséges;
- az automatikus adatkonverzióhoz szükséges még engedélyezni a fordító számára az `OverloadedStrings`, `DriveGeneric`, illetve `DeriveAnyClass` kiterjesztéseket:

```
{-# LANGUAGE OverloadedStrings, DeriveGeneric, DeriveAnyClass #-}
```

JSON formátumú adatok

- A következő két slide-on található kódsor egy `Szemely` típusú értéket alakít át JSON formába, majd elvégzi a visszaalakítást:

```
{-# LANGUAGE OverloadedStrings, DeriveGeneric #-}
import Data.Aeson
import Data.Maybe
import GHC.Generics

data Szemely = Szemely {
       nevSz :: String,
       szEvSz :: Int
   } deriving (Show, Generic)

instance ToJSON Szemely
instance FromJSON Szemely
```

JSON formátumú adatok

```
foJSON = do
   let lsJ = encode $ Szemely {
       nevSz = "Kiss Zsuzsa",
       szEvSz = 2000}
   putStrLn $ "JSON formaban: " ++ show lsJ
   let lsD = decode lsJ :: Maybe Szemely
   putStrLn $ "dekodolva: " ++ show (fromJust lsD)

> foJSON
JSON formaban: "{\"szEvSz\":\"2000\",\"nevSz\":Kiss Zsuzsa\}"
dekodolva: Szemely {nevSz = "Kiss Zsuzsa", szEvSz = 2000}
```

JSON formátumú adatok

1. feladat

A `szemelyek.json` állomány JSON szerkezetű. Írjunk egy Haskell programot, amely dekodólja az állományban levő adatokat, átalakítja a JSON típusú adatokat `Szemelyek` típusú adatokká, rendezi őket születési év alapján, majd a rendezett listát visszaalakítja JSON formába, és kiírja az eredményt a `szemelyekR.json` állományba.

Legyen a `szemelyek.json` állomány tartalma a következő:

```
{"szemelyek":[
   {"nevSz"          :"Nagy Zsuzsa"
   ,"szEvSz"         :2001
   },

   {"nevSz"          :"Kiss Tibor"
   ,"szEvSz"         :1999
   },

   {"nevSz"          :"Szasz Zsombor"
   ,"szEvSz"         :2002
   },

   {"nevSz"          :"Szep Attila"
   ,"szEvSz"         :1998
   }
]}
```

JSON formátumú adatok

```
{-# LANGUAGE OverloadedStrings, DeriveGeneric, DeriveAnyClass #-}
import Data.Aeson ( decode, encode, FromJSON, ToJSON )
import Data.Aeson.Encode.Pretty ( encodePretty )
import GHC.Generics ( Generic )
import qualified Data.ByteString.Lazy.Char8 as L8
import Data.List ( sortOn, sortBy )
import Data.Ord ( comparing )

data Szemely = Szemely {
        nevSz :: String,
        szEvSz :: Int
    } deriving (Show, Generic, Read, ToJSON, FromJSON)

data Szemelyek = Szemelyek {
        szemelyek :: [Szemely]
    } deriving (Show, Generic, Read, ToJSON, FromJSON)
```

JSON formátumú adatok

- a `foSzemely`-ben `readFile`-lal beolvassuk az adatokat, majd `decode`-dal átalakítjuk a JSON formátumú adatokat `Maybe Szemelyek` típussá
- az adatokat a `sortBy`-al, vagy a `sortOn`-al a `szEv` mező alapján rendezzük
- az `encodePretty`-vel végezzük a visszaalakítást, azért, hogy az új sor jelek és tabulátorok segítségével tudjuk megoldani az adatok elegáns formázását
- ahhoz, hogy az adatok konvertálása hibamentesen történjen, fontos, hogy megtartsuk a `Szemely` és `Szemelyek` típusdeklarációk esetében a `nev`, `szEv` és `szemelyek` megnevezéseket: válasszunk olyan mezőneveket, amelyek a JSON formátumban levő állományban használatosak

```
foSzemely :: IO ()
foSzemely = do
    temp <- L8.readFile "szemelyek.json"
    let Just jTemp = decode temp :: Maybe Szemelyek
    let ls = szemelyek jTemp
    --let sLs = sortBy (comparing szEvSz) ls
    let sLs = sortOn szEvSz ls
    let jLs = encodePretty (Szemelyek sLs)
    L8.writeFile "szemelyekR.json" jLs
```

JSON formátumú adatok

2. feladat

A `tudosok.json` állomány JSON szerkezetű, tudósok adatait tárolja: vezetéknév, nemzetiség, születési év és elhalálozási év. Írjunk egy Haskell-programot, amely meghatározza az állományban szereplő tudósok születési év szerint rendezett sorrendjét, illetve egy másik opció választása esetén az életkor szerinti rendezett sorrendet írja ki. Abban az esetben, ha nem jelenik meg az elhalálozási év, az életkor helyett írja ki a `kortars` szót.

Egy ilyen szerkezetű állomány letölthető a következő linkről:
https://ms.sapientia.ro/~mgyongyi/Funk_Log/Jegyzet/tudosok.json

A szükséges kiterjesztések a következők:

```
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE DeriveGeneric #-}
{-# LANGUAGE DeriveAnyClass #-}
```

JSON formátumú adatok

A szükséges importok a következők:

```
import qualified Data.ByteString.Lazy.Char8 as L8
import Data.Aeson
import GHC.Generics
import Data.List
import Data.Ord
```

Két adatszerkezetet definiálunk:

```
data Tudos = Tudos {
    nev :: String,
    nemzetiseg :: String,
    szEv :: Int,
    hEv :: Maybe Int
    } deriving (Show, Generic, FromJSON)

data Tudosok = Tudosok {
    tudosok :: [Tudos]
    } deriving (Show, Generic, FromJSON)
```

JSON formátumú adatok

- a főfüggvény a `foTudos1`, ahol a rendezésekhez a `sortBy`-ot használjuk
- az `auxEv` a kiíratáshoz szükséges karakterláncot határozza meg
- a `korF`, ha létezik elhalálozási év, akkor kiszámítja az életkor értékét, ellenkező esetben `-1` lesz a függvény kimenete
- a `myPrint` az adatok formázott kiíratását végzi

```
foTudos1 = do
   inf <- L8.readFile "tudosok.json"
   let temp = decode inf :: Maybe Tudosok
   case temp of
       Nothing -> print "JSON adatfeldolgozasi hiba"
       Just tLs -> do
           putStrLn "(szEv/eKor)?"
           str <- getLine
           let ls = tudosok tLs
           case str of
               "szEv" -> do
                   putStrLn "rendezve:"
                   let sLs = sortBy (comparing szEv) ls
                   mapM_ (putStrLn . auxEv) sLs
               "eKor" -> do
                   putStrLn "rendezve: "
                   let sLs = sortBy (comparing korF) ls
                   mapM_ myPrint sLs
               _ -> putStrLn "hibas bemenet!"
```

JSON formátumú adatok

```
auxEv :: Tudos -> String
auxEv ls = nev ls ++ ", szuletesi ev: " ++ (show . szEv) ls

korF :: Tudos -> Int
korF x = case hEv x of
    Just xHEv -> xHEv - szEv x
    Nothing -> -1

myPrint :: Tudos -> IO()
myPrint ls
    | kF == -1 = do
        let rLs = nev ls ++ " " ++ nemzetiseg ls ++ " kortars"
        putStrLn rLs
    | otherwise = do
        let rLs = nev ls ++ " " ++ nemzetiseg ls ++ " " ++ show kF
        putStrLn rLs
    where
    kF = korF ls
```

JSON formátumú adatok

3. feladat

Írjunk egy Haskell-programot, amely kiírja a képernyőre adott nemzetiségű tudósok listáját, ahol az adatokat az előző feladatban megadott `tudosok.json` állományban találjuk. Használjuk a megadott `Tudos` és `Tudosok` adatszerkezeteket.

```
foTudos2 = do
    inf <- L8.readFile "tudosok.json"
    let temp = decode inf :: Maybe Tudosok
    case temp of
        Nothing -> print "JSON adatfeldolgozasi hiba"
        Just tLs -> do
            let ls = tudosok tLs
            let hLs = halmazL ls
            mapM_ (\k -> putStr $ k ++ " ") hLs
            putStr "\nnemzetiseg: "
            temp <- getLine
            putStrLn $ "\n" ++ temp ++ " tudosok: "
            mapM_ putStrLn $ valogat temp ls
```

JSON formátumú adatok

- a `halmazL` függvény meghatározza az állományban szereplő különböző nemzetiségeket
- a megfelelő nemzetiségű tudósok kiválogatását a `valogat` végzi, amely a válogatás mellett létrehoz egy `String` típusú értékekből álló listát, ahol egy `String` típusú értéket a megfelelő nemzetiségű tudós `nev` és `szEv` mezőjéből építjük fel

```
halmazL :: [Tudos] ->[String]
halmazL [] = []
halmazL (k : ve) = nemzetiseg k : halmazL [x | x <- ve,
                   nemzetiseg k /= nemzetiseg x]

valogat :: String -> [Tudos] -> [String]
valogat nemz ls = [nev x ++ " " ++ show (szEv x) |
                   x <- ls, nemzetiseg x == nemz]
```

JSON formátumú adatok

4. feladat

Írjunk egy Haskell-programot, amely létrehozza megadott nemzetiségű tudósok listáját, majd a létrehozott lista alapján JSON formátumban kiírja az adatokat egy állományba, ahol az adatokat az előző feladatban megadott `tudosok.json` állományban találjuk.

- ugyanazokat a kitejesztéseket kell engedélyezni, mint az előző példánál az importok listájában, azonban szerepelnie kell az `Encode.Pretty`-nek, illetve `Data.Char`-nak
- a `Tudos` és `Tudosok` típusdeklarációkat is módosítani kell, mert a származtatásnál fel kellett tüntetni a `ToJSON` típusosztályt:

```
...
import Data.Aeson.Encode.Pretty
import Data.Char

data Tudos = Tudos {
    nev :: String,
    nemzetiseg :: String,
    szEv :: Int,
    hEv :: Maybe Int
    } deriving (Show, Generic, FromJSON, ToJSON)
```

JSON formátumú adatok

```
data Tudosok = Tudosok {
    tudosok :: [Tudos]
    } deriving (Show, Generic, FromJSON, ToJSON)
```

- a `foTudos3` az előző példánál megadott `halmazL` függvényt használja
- a `valogatF` függvény hasonló elgondoláson alapszik, mint az előző példánál megadott `valogat` függvény, csak a listaelemek most `Tudos` típusúak lesznek
- az `uTemp`-be létrehozzuk a beolvasott nemzetiség nagybetűvel kezdődő alakját, mert ezt a karakterláncot használva alakítjuk ki az állomány nevét, amelybe a kért nemzetiségű tudósokat kiírjuk

JSON formátumú adatok

```
foTudos3 = do
    inf <- L8.readFile "tudosok.json"
    let temp = decode inf :: Maybe Tudosok
    case temp of
        Nothing -> print "JSON adatfeldolgozasi hiba"
        Just tLs -> do
            let ls = tudosok tLs
            let nLs = halmazL ls
            mapM_ (\k -> putStr $ k ++ " ") nLs
            putStr "\nnemzetiseg: "
            temp <- getLine
            let nLs = valogatF temp ls
            let jLs = encodePretty (Tudosok nLs)
            let uTemp = toUpper (head temp) : tail temp
            L8.writeFile ("tudosok" ++ uTemp ++ ".json") jLs

valogatF :: String -> [Tudos] -> [Tudos]
valogatF nemz = filter (\x -> nemzetiseg x == nemz)
```