Laboratorul 13: I/O

Operații de intrare/ieșire în Haskell

```
import Data.Char
```

Vom începe prin a exersa operațiile de citire și scriere.

Exemplul 1

Citirea de la tastatura a unui șir și afișarea rezultatului obținut după prelucrare.

```
prelStr strin = map toUpper strin

ioString = do
           strin <- getLine
           putStrLn $ "Intrare\n" ++ strin
           let  strout = prelStr strin
           putStrLn $ "Iesire\n" ++ strout
```

Exemplul 2

Citirea de la tastatura a unui număr și afișarea rezultatului obținut după prelucrare.

```
prelNo noin = sqrt noin

ioNumber = do
           noin <- readLn :: IO Double
           putStrLn $ "Intrare\n" ++ (show noin)
           let  noout = prelNo noin
           putStrLn $ "Iesire"
           print noout
```

Observati utilizarea functiilor “readLn”, “show” si “print”.

Exemplul 3

Citirea din fișier de intrare și afișarea rezultatului într-un fișier de ieșire.

```haskell
inoutFile = do
              sin <- readFile "Input.txt"
              putStrLn $ "Intrare\n" ++ sin
              let sout = prelStr sin
              putStrLn $ "Iesire\n" ++ sout
              writeFile "Output.txt" sout
```

Atenție! Funcția `readFile` întoarce un rezultat de tipul `IO String`.

Exercițiul 1

Scrieți un program care citește de la tastatură un număr n și o secvență de n persoane, pentru fiecare persoană citind numele si varsta. Programul trebuie sa afiseze persoana (sau persoanele) cu varsta cea mai mare. Presupunem ca varsta este exprimata printr-un `Int`.

Exemplu de intrare:

```
3
Ion Ionel Ionescu
70
Gica Petrescu
99
Mustafa ben Muhamad
7
```

Exemplu de iesire:

```
Cel mai in varsta este Gica Petrescu (99 ani).
```

Exercițiul 2

Aceeași cerință ca mai sus, dar datele se citesc dintr-un fișier de intrare, în care fiecare linie conține informația asociată unei persoane, numele si varsta fiind separate prin virgulă (vedeți fișierul `ex2.in`).

Indicație: pentru a separa numele de varsta puteti folosi funcția `splitOn` din modulul `Data.List.Split`.

Exercițiul 3

Citiți capitolul I/O din M. Lipovaca, Learn You a Haskell for Great Good! http://learnyouahaskell.com/input-and-output

Propria noastră monadă I/O

O acțiune de întrare ieșire care produce un rezultat de tip `a` poate fi modelată ca o funcție de la un șir de intrare la un tuplu format dintr-un element de tip

a reprezentând rezultatul, un șir de caractere reprezentând partea din șirul de intrare care nu a fost consumată și un șir de caractere reprezentând ce afișează acțiunea.

```
type Input = String
type Output = String

newtype MyIO a = MyIO { runIO :: Input -> (a, Input, Output)}
```

Exercițiul 1

Definiți functiile de bază `myGetChar` și `myPutChar`:

```
myGetChar :: MyIO Char
myGetChar = undefined

testMyGetChar :: Bool
testMyGetChar = runIO myGetChar "Ana" == ('A', "na", "")

myPutChar :: Char -> MyIO ()
myPutChar = undefined

testMyPutChar :: Bool
testMyPutChar = runIO (myPutChar 'C') "Ana" == ((), "Ana", "C")
```

Exercițiul 2

Faceți `MyIO` instanță a clasei de tipuri `Functor`:

```
instance Functor MyIO where

testFunctorMyIO :: Bool
testFunctorMyIO = runIO (fmap toUpper myGetChar) "ana" == ('A', "na", "")
```

Exercițiul 3

Faceți `MyIO` instanță a clasei de tipuri `Applicative`. Observație: când propagăm efectele laterale, șirul de intrare se consumă, iar șirurile de ieșire se concatenează.

```
instance Applicative MyIO where

testPureMyIO :: Bool
testPureMyIO = runIO (pure 'C') "Ana" == ('C', "Ana", "")

testApMyIO :: Bool
testApMyIO = runIO (pure (<) <*> myGetChar <*> myGetChar) "Ana" == (True, "a", "")
```

Exercițiul 4

Faceți `MyIO` instanță a clasei de tipuri `Monad`.

```
instance Monad MyIO where

testBindMyIO :: Bool
testBindMyIO = runIO (myGetChar >>= myPutChar) "Ana" == ((), "na", "A")
```