

# Monada IO.

## Laborator 9

Reamintiți că `IO a` este un tip ale cărui valori sunt *acțiuni care, după ce sunt executate, produc o valoare de tip a*.

De exemplu, funcția `putStrLn :: String -> IO ()` primește un șir de caractere și întoarce o acțiune, care, după ce este executată, produce o valoare de tip *unit* (tipul *unit* este notat cu `()`).

**Exercițiul 0.1.** Care este singura valoare de tip `()`?

Pentru a face un program executabil, este suficient să scrieți o funcție cu numele `main`, care să nu primească niciun argument, și care să întoarcă o valoare de tip `IO ()`, adică o acțiune.

```
main :: IO ()
main = putStrLn "Hello, World!"
```

Dacă compilați un fișier care conține o funcție `main`, la rularea fișierului executabil, Haskell va evalua funcția `main` (rezultatul va fi o acțiune) și apoi va *executa acțiunea*.

```
$ ghc fisier.hs -o fisierexecutabil
...
$ ./fisierexecutabil
Hello, World!
```

Puteți folosi operatorul binar `(>>) :: IO a -> IO b -> IO b` pentru a *compune secvențial două acțiuni*.

```
main :: IO ()
main = (putStrLn "Hello, World!") >> (putStrLn "All good.")
```

**Exercițiul 0.2.** Folosiți `>>` pentru a crea o secvență de trei acțiuni.

Folosiți comanda `:info` pentru a afla dacă parserul pentru `>>` asociază implicit la stânga sau la dreapta.

Acțiunile întoarse de funcția `putStrLn` produc, după execuție, o valoare de tip `unit`, care nu este foarte interesantă. Alte acțiuni, cum ar fi acțiunea întoarsă de funcția `getLine`, produc valori interesante:

```
> :t getLine
getLine :: IO String
```

Când este executată, acțiunea întoarsă de funcția `getLine` citește de la consolă un șir de caractere. Iată un exemplu de utilizare a funcției `getLine`:

```
main :: IO ()
main = (putStrLn "What is your name?") >>
      (getLine) >>
      (putStrLn "Hello, ...!")
```

Într-o secvență de acțiuni, cum putem să punem mâna pe valoarea produsă de una dintre acțiuni? Nu putem dacă folosim doar operatorul (>>):

```
:t (>>) :: IO a -> IO b -> IO b
(>>) :: IO a -> IO b -> IO b :: IO a -> IO b -> IO b
```

După cum observați, valoare de tip `a` produsă de prima acțiune *se pierde*. Dacă avem nevoie de ea, trebuie să folosim operatorul >>=:

```
:t (>>=) :: IO a -> (a -> IO b) -> IO b
(>>=) :: IO a -> (a -> IO b) -> IO b :: IO a -> (a -> IO b) -> IO b
```

Acesta acceptă două argumente:

1. primul argument (de tip `IO a`) este o acțiune care, dacă este executată, produce o valoare de tip `a`;
2. al doilea argument (de tip `a -> IO b`) este o funcție care primește o valoare de tip `a` și întoarce o acțiune care, dacă este executată, produce o valoare de tip `b`.

Operatorul >>= întoarce o acțiune care, dacă este executată, execută întâi acțiunea de tip `IO a` și folosește valoarea produsă ca argument al funcției de tip `a -> IO b`, iar apoi execută acțiunea întoarsă de această funcție. Iată un exemplu:

```
main :: IO ()
main = (putStrLn "What is your name?") >>
      (getLine >>=
       (\name -> putStrLn ("Hello, " ++ name ++ "!")))
```

**Exercițiul 0.3.** Scrieți un program care citește de la tastatură prenumele utilizatorului, apoi numele acestuia, și afișează un mesaj de întâmpinare (indicație: veți folosi de două ori >>=).

Putem rescrie programul de mai sus folosind notația `do`:

```
main :: IO ()
main = do
  putStrLn "What is your name?"
  name <- getLine
  putStrLn ("Hello, " ++ name ++ "!")
```

**Exercițiul 0.4.** Refaceți programul de mai sus cu nume și prenume folosind notația `do`.

Pentru a repeta în buclă o acțiune, putem folosi recursia:

```
main :: IO ()
main = do
  putStrLn "What is your name?"
  name <- getLine
  putStrLn ("Hello, " ++ name ++ "!")
  main
```

Pentru a ieși din programul de mai sus, folosiți Ctrl-C sau Ctrl-Z, în funcție de sistemul de operare.

**Exercițiul 0.5.** Scrieți programul recursiv de mai sus folosind `>>` și `>>=`, fără notația `do`.

**Exercițiul 0.6.** Modificați programul de mai sus astfel încât să ceară (la infinit) prenumele și apoi numele.

Funcția `return :: a -> IO a` primește la intrare o valoare și întoarce o acțiune care, când este executată, nu face nimic și produce valoarea dată ca argument.

```
main :: IO ()
main = return ()
```

**Atenție!** Funcția `return` nu seamănă deloc cu cuvântul cheie `return` din limbajele din familia C(++)/Java. Pentru a înțelege acest lucru, explicați comportamentul următoarelor programe:

```
main :: IO ()
main = do
  putStrLn "What is your name?"
  name <- return "Victor"
  putStrLn ("Hello, " ++ name ++ "!")
```

```
main :: IO ()
main = do
  putStrLn "What is your name?"
  return ()
  name <- getLine
  putStrLn ("Hello, " ++ name ++ "!")
```

Pentru a evita bucla infinită la Exercițiile ?? și ??, este suficient să introducem o condiție de terminare:

```
main :: IO ()
main = do
  putStrLn "What is your name?"
  name <- getLine
  putStrLn ("Hello, " ++ name ++ "!")
  if name == "" then
    return ()
  else
    main
```

**Exercițiul 0.7.** Modificați programul cu citirea prenumelor și a numelor astfel încât să se oprească când prenumele **sau** numele sunt șirul vid.

**Exercițiul 0.8.** Scrieți un program care citește de la tastatură șiruri de caractere și afișează aceste șiruri în UPPERCASE.

**Exercițiul 0.9.** Căutați următoarele funcții în documentația bibliotecii standard (inclusiv în ce modul se află) și scrieți tipul lor:

1. `openFile :: ...`

2. `hGetContents :: ...`
3. `hGetLine :: ...`
4. `hClose :: ...`
5. `getArgs :: ...`
6. `getProgName :: ...`
7. `hPutStr :: ...`

**Exercițiul 0.10.** Scrieți un program care afișează conținutul fișierului text “exemplu.txt” pe ecran.

**Exercițiul 0.11.** Modificați programul astfel încât numele fișierului să fie primul argument din linia de comandă.

**Exercițiul 0.12.** Modificați programul astfel încât, dacă linia de comandă nu este corectă, să afișeze sintaxa de apel a programului (folosiți funcția `getProgName`).

**Exercițiul 0.13.** Modificați programul astfel încât să afișeze conținutul fișierului în UPPER-CASE.

**Exercițiul 0.14.** Creați un program care ghicește un număr între 1 și 100 (folosind căutare binară). Iată un exemplu de interacțiune:

```
$ ghiceste
Numarul de ghicit este >= 50?
Nu
Numarul de ghicit este >= 25?
Nu
Numarul de ghicit este >= 12?
Nu
Numarul de ghicit este >= 6?
Da
Numarul de ghicit este >= 9?
asdf
Nu am înțeles. Numarul de ghicit este >= 9?
Nu
Numarul de ghicit este >= 7?
Da
Numarul de ghicit este >= 8?
Nu
Atunci numărul este 7.
```

**Exercițiul 0.15.** Explicați de ce este crucială evaluarea leneșă pentru execuția acțiunilor IO.

**Exercițiul 0.16.** Proiectați un experiment prin care să demonstrați că funcția `hGetContents` nu citește tot conținutul unui fișier, decât dacă este strict necesar.

**Exercițiul 0.17.** Scrieți un program Haskell care citește un număr `n` de la intrarea standard și afișează `n` numere pseudoaleatorii.