Входно-изходни операции в Haskell – част 2

Трифон Трифонов

Функционално програмиране, 2025/26 г.

8 януари 2026 г.

Управляващи функции

Можем да работим с трансформации с функции от по-висок ред:

- `import Control.Monad`
- `sequence :: [IO a] -> IO [a]`
  - композира трансформации и събира резултатите им в списък
  - `getInts = sequence . (`replicate` getInt)`
- `mapM :: (a -> IO b) -> [a] -> IO [b]`
  - композира списък от трансформации по списък от стойности
  - `mapM = sequence . map`
  - `printRead s = do putStr $ s ++ " = "; getInt`
  - `readCoordinates = mapM printRead ["x", "y", "z"]`
- `mapM_ :: (a -> IO b) -> [a] -> IO ()`
  - Също като `mapM`, но изхвърля резултата
  - `printList = mapM_ print`
- `forever :: IO a -> IO b`
  - безкрайна композиция на една и съща трансформация (както `repeat` за списъци)
  - `forever $ do line <- getLine; putStrLn line`

Средно аритметично на числа v2.0

```
readInt :: String -> IO Int
readInt s = do putStr $ "Моля, въведете " ++ s ++ ": "
               getInt

findAverage :: IO Double
findAverage = do n <- readInt "брой"
                 l <- mapM (readInt.("число #"++).show) [1..n]
                 let s = sum l
                 return $ fromIntegral s / fromIntegral n

main = forever $
       do avg <- findAverage
          putStrLn $ "Средното аритметично е: " ++ show avg
          putStrLn "Хайде отново!"
```

Монади

- `IO` е пример за монада
- Монадите са конструкции, които „опаковат“ обекти от даден тип
- **Примери:**
  - `IO` опакова стойност във входно/изходна трансформация
  - `Maybe` опакова стойност с „флаг“ дали стойността съществува
  - `[a]` опакова няколко „алтернативни“ стойности в едно
  - `r -> a` опакова стойност от тип `a` в „машинка“, която я пресмята при подаден параметър от тип `r`
  - `s -> (a,s)` опакова стойност от тип `a` в „действие“, което променя дадено състояние от тип `s`

Монадни операции

- `Monad` е клас от типови конструктори, които са монади
- „Опаковката“ понякога е прозрачна... (пример: `Maybe`, `[a]`)
- ... но често е еднопосочна: един път опакована, не можем да извадим стойността извън опаковката... (пример: `IO`, `r -> a`)
- ... но можем да я преопаковаме!
- `(>>=) :: Monad m => m a -> (a -> m b) -> m b`
- оператор за „свързване“ на опаковани стойности
- `b = a >>= f`:
  - поглеждаме стойността `x` в опаковката `a`
  - прилагаме функцията `f` над `x`
  - и получаваме нова опакована стойност `b`

Императивен стил чрез монади

- do всъщност е синтактична захар за поредица от „свързвания“
- **Примери:**

```
main = do line <- getLine
          putStrLn $ "Въведохте: " ++ line

main = getLine >>= putStrLn . ("Въведохте: " ++)

findAverage = do putStr "Моля, въведете брой: "
                 n <- getInt
                 s <- readAndSum n
                 return $ fromIntegral s / fromIntegral n

findAverage = putStr "Моля, въведете брой: " >>=
              (\_ -> getInt >>=
              (\n -> readAndSum n >>=
```