

Języki i paradygmaty programowania (Info, III rok) 16/17

Kokpit ► Moje kursy ► JiPP.INFO.III.16/17 ► T3 13-19.3 Monady 1 ► Lab Haskell 2

Lab Haskell 2

Zadanie 1

Rozważmy typ drzew trochę inny niż na wykładzie

```
data Tree a = Empty | Node a (Tree a) (Tree a) deriving (Eq, Ord, Show)
```

a. stwórz własne instancje `Eq`, `Show`

```
instance Show a => Show (Tree a) where
    show t = ...

instance Eq a => Eq (Tree a) where
    t1 == t2 = ...
```

b. skonstruuj instancje klasy `Functor`:

```
-- class Functor f where
--     fmap :: (a -> b) -> f a -> f b

instance Functor Tree where...

*Tree> fmap (+1) $ Node 1 Empty Empty
Node 2 Empty Empty
```

c. Napisz funkcje

```
toList :: Tree a -> [a]
```

która zamieni drzewo w liste elementow drzewa (w porzadku infiksowym)

d. zaimplementuj drzewa BST z funkcjami

```
insert :: (Ord a) => a -> Tree a -> Tree a
contains :: (Ord a) -> a -> Tree a -> Bool
fromList :: (Ord a) => [a] -> Tree a
```

e. Stwórz modul drzew BST i wykorzystaj go w innym module do sortowania `[Int]`

- przy pomocy ghci
- przy pomocy ghc --make

Zadanie 2

a. Uzupełnij instancje klasy Functor dla Either e:

```
import Prelude hiding(Either(..))
data Either a b = Left a | Right b

instance Functor (Either e) where
  -- fmap :: (a -> b) -> Either e a -> Either e b
```

b. napisz funkcje

```
reverseRight :: Either e [a] -> Either e [a]
```

odwracajaca liste zawarta w Right (dwie wersje, bezposrednio i z uzyciem fmap)

c. Zdefiniuj klasę Pointed (funkcyjnych pojemników z singletonem)

```
class Functor f => Pointed f where
  pure :: a -> f a
```

i jej instancje dla list, Maybe, Tree

Zadanie 3

Importy i moduły biblioteczne, np. przećwiczyć <http://learnyouahaskell.com/modules>

a. Napisz funkcję

```
readInts :: String -> [Int]
```

która odczyta z napisu występujące w nim liczby naturalne, np

```
*Main> readInts "1 23 456 7.8 abc 9"
[1,23,456,9]
*Main> readInts "foo"
[]
```

użyj funkcji `isDigit` z modułu `Data.Char` oraz funkcji `map`, `filter`, `all` z `Prelude`

b. Napisz podobną funkcję `readInts2 :: String -> Either String [Int]` która da listę liczb, jeśli wszystkie słowa jej argumentu są liczbami a komunikat o błędzie w przeciwnym przypadku

Może się przydać funkcja `reverseRight` z zad. 3b (lub `fmap` dla `Either`)

```
*Main> readInts2 "1 23 456 foo 9"
Left "Not a number: foo"
*Main> readInts2 "1 23 456"
Right [1,23,456]
```

c. Napisz funkcję

```
sumInts :: String -> String
```

- jeśli wszystkie słowa jej argumentu są liczbami da reprezentację ich sumy
- wpp komunikat o błędzie

stwórz program uruchamiający funkcję sumInts przy pomocy interact.

Zadanie 4

Rozważmy typ dla wyrażeń arytmetycznych z let:

```
data Exp
  = EInt Int           -- stała całkowita
  | EAdd Exp Exp       -- e1 + e2
  | ESub Exp Exp       -- e1 - e2
  | EMul Exp Exp       -- e1 * e2
  | EVar String        -- zmienna
  | ELet String Exp Exp -- let var = e1 in e2
```

a. Napisz instancje Eq oraz Show dla Exp

b. Napisz instancje Num dla Exp tak, żeby można było napisać

```
testExp2 :: Exp
testExp2 = (2 + 2) * 3
```

(metody abs i signum mogą mieć wartość undefined)

c. Napisz funkcję `simpl`, przekształcającą wyrażenie na równoważne prostsze wyrażenie np.

$0x + 1y \rightarrow y$

(nie ma tu precyzyjnej specyfikacji, należy użyć zdrowego rozsądku; uwaga na zapętlenie).

d. Można dopisać liczenie pochodnej względem danej zmiennej:

```
deriv :: String -> Exp -> Exp
```

Zadanie 6 (opcjonalne, dla znudzonych)

Zdefiniuj klasę

```
infixl 4 <*>
class Pointed f => Applicative f where
  (<*>) :: f(a->b) -> f a -> f b
```

i jej instancje dla `Maybe`, `[]`, tak aby spełnione były zależności

```
pure id <*> v = v

pure (.) <*> u <*> v <*> w = u <*> (v <*> w)

pure f <*> pure x = pure (f x)

u <*> pure y = pure ($ y) <*> u
```

Jaka będzie wartość wyrażenia

$[(+1), (*2)] <*> [1, 2]$

Ostatnia modyfikacja: czwartek, 19 marzec 2015, 09:12

NAWIGACJA



Kokpit

- Strona główna

Strony

Moje kursy


JNPI.INFO.II.16/17

JiPP.INFO.III.16/17

Uczestnicy

 Odznaki

 Kompetencje

 Oceny


Główne składowe


T1 27.2-5.3 Haskell 1

T2 6-12.3 Haskell 2

T3 13-19.3 Monady 1

 **Lab Haskell 2**

 Wykład 3: monady 1

 Zadanie 1 - Haskell

 Opis zadania w PDF

 TestAuto.hs

 SimpleTests.hs

20-26.3 Monady 2

27.3-2.4 Składnia 1

3.4-9.4

10.4-23.4

24-30.4

1.5-14.5

15-21.5

22.5-28.5

29.5-4.6

5.6-11.6

12-14.6

Bonus

TOPI*.MAT.16/17

ADMINISTRACJA



Administracja kursem