

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

Kokpit ► Moje kursy ► JiPP.INFO.III.16/17 ► 20-26.3 Monady 2 ► Lab Monady 1

Lab Monady 1

Zadanie 1

W poprzednim tygodniu pisaliśmy funkcje

```
readInts2 :: String -> Either String [Int]
sumInts :: String -> String
```

Przepisz funkcje readInts2, sumInts na notację monadyczną. Można uzyć readMaybe lub readEither.

(Uwaga: w tym zadaniu raczej nadal tworzymy funkcje String -> String i korzystamy z interact niż bezpośrednio z IO, chyba że ktoś bardzo chce)

Zadanie 2 - IO

- a. Napisz program który wypisze swoje argumenty, każdy w osobnej linii
- b. Napisz program, który będzie pytał użytkownika o ulubiony język programowania, tak długo aż odpowiedzią będzie 'Haskell' ;)
- c. Napisz uproszczoną wersję programu wc (wypisującą ilość linii, słów i znaków w pliku o nazwie podanej jako argument)

Zadanie 3.

Uzupełnić przykład z wykładu:

```
data ParseError = Err {location::Int, reason::String}
instance Error ParseError ...
type ParseMonad = Either ParseError
parseHexDigit :: Char -> Int -> ParseMonad Integer
parseHex :: String -> ParseMonad Integer
toString :: Integer -> ParseMonad String

-- convert zamienia napis z liczba szesnastkowa
-- na napis z liczba dziesietna
convert :: String -> String
convert s = str where
(Right str) = tryParse s `catchError` printError
tryParse s = do {n <- parseHex s; toString n}
printError e = ...</pre>
```

Zadanie 4. Operacje monadyczne

Napisz własną implementację funkcji

```
sequence :: Monad m => [m a] -> m [a]
mapM :: Monad m => (a -> m b) -> [a] -> m [b]
forM :: Monad m => [a] -> (a -> m b) -> m [b]
```

Zadanie 5. (opcjonalne)

Nieco inny od monad model obliczeń reprezentuje klasa Applicative :

```
class Functor f => Applicative f where
pure :: a -> f a
(<*>) :: f(a->b) -> f a -> f b
```

- pure to to samo co return
- Operator (<*>) reprezentuje sekwencjonowanie obliczeń podobne do (=<<) z tym, że kolejne obliczenie nie zależy od wyniku poprzedniego (choć jego wynik oczywiście może).
- Dla każdej monady można zdefiniować instancję Applicative:

```
pure = return
mf <*> ma = do { f <- mf; a <- ma; return mf ma }</pre>
```

a. Zdefiniuj instancje Applicative dla Maybe i Either b. Zdefiniuj operację *> będącą analogiem >> (czyli ignorującą wynik pierwszego obliczenia):

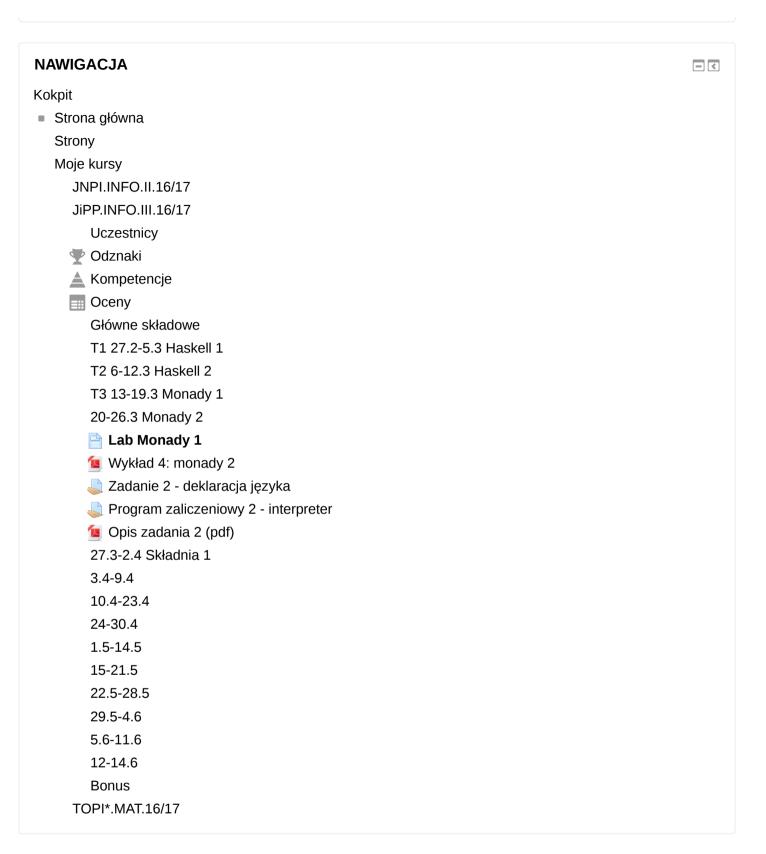
```
(*>) :: f a -> f b -> f b
```

c. Zdefiniuj analogiczną operację ignorującą wynik drugiego obliczenia:

```
(<*) :: fa -> fb -> fa
```

d. Spróbuj wykonać zadanie 3 używając | Applicative | zamiast | Monad

Ostatnia modyfikacja: czwartek, 17 marzec 2016, 07:59



ADMINISTRACJA

Administracja kursem