

# Metody programowania 2017

## Lista zadań nr 9

Na zajęcia 9 i 10 maja 2017

Niech

```
reverse, rev :: [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
rev = aux [] where
  aux ys [] = ys
  aux ys (x:xs) = aux (x:ys) xs
```

**Zadanie 1 (1 pkt).** Pokaż, że  $\text{reverse} = \text{rev}$ .

**Zadanie 2 (1 pkt).** Pokaż, że dla dowolnej skończonej listy  $xs$  zachodzi równość:

$$\text{reverse}(\text{reverse } xs) = xs$$

*Wskazówka:* udowodnij w pierw odpowiedni lemat o funkcji  $++$ . Wskaż miejsce w którym dowód twierdzenia bez założenia o skończoności listy  $xs$  zaczyna się.

**Zadanie 3 (1 pkt).** Udowodnij, że dla dowolnej częściowej listy  $xs$  mamy:

$$xs ++ ys = xs$$

gdzie  $ys$  jest dowolną listą. Pokaż, gdzie dowód zaczyna się dla list skończonych.

**Zadanie 4 (1 pkt).** Na czym polega problem w poniższej definicji funkcji Fibonacciego:

```
fib :: Integer -> Integer
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Zaprogramuj funkcję która wywołana z parametrem  $n$  wykonuje  $O(n)$  iteracji w celu obliczenia wartości  $\text{fib } n$ .

**Zadanie 5 (1 pkt).** Zaprogramuj w Haskellu funkcję

```
roots :: (Double, Double, Double) -> [Double]
```

wyznaczającą listę miejsc zerowych trójmianu kwadratowego o podanych współczynnikach. *Wskazówka:* typ `Double` należy do klasy `Ord`, zdefiniowano więc dla niego metodę

```
compare :: Double -> Double -> Ordering
```

gdzie

```
data Ordering = LT | EQ | GT
deriving (Eq, Ord, Enum, Read, Show, Bounded)
```

W preludium standardowym zdefiniowano też funkcję

```
sqrt :: (Floating a) => a -> a
```

a typ `Double` należy do klasy `Floating`. Niech

```
data Roots = No
           | One Double
           | Two (Double, Double)
           deriving Show
roots :: (Double, Double, Double) -> Roots
```

Zaprogramuj tę funkcję. Czy jest lepsza niż poprzednia? Podaj argumenty za i przeciw. Skomentuj następnie funkcję o sygnaturach:

```
roots :: Double -> Double -> Double -> [Double]
roots :: [Double] -> [Double]
```

**Zadanie 6 (1 pkt).** Nie korzystając z faktu, że typ `Integer` należy do klasy `Show` zaprogramuj funkcję

```
integerToString :: Integer -> String
```

*Wskazówka:* wykorzystaj funkcję

```
unfoldr :: (b -> Maybe (a, b)) -> b -> [a]
```

z modułu `List` daną wzorem

```
unfoldr f b =
  case f b of
    Nothing -> []
    Just (a,b) -> a : unfoldr f b
```

gdzie

```
data Maybe a = Nothing
             | Just a
             deriving (Eq, Ord, Read, Show)
```

Moduł `Char` udostępnia funkcję

```
intToDigit :: Int -> Char
```

preludium standardowe oferuje następującą metodę klasy `Enum`:

```
fromEnum :: (Enum a) => a -> Int
```

a typ `Integer` należy do klasy `Enum`.

**Zadanie 7 (1 pkt).** Zbiory, także nieskończone, można reprezentować w postaci ich funkcji charakterystycznych:

```
newtype FSet a = FSet (a -> Bool)
```

Zdefiniuj następujące operacje:

```
empty :: FSet a
singleton :: Ord a => a -> FSet a
fromList :: Ord a => [a] -> FSet a
union :: Ord a => FSet a -> FSet a -> FSet a
intersection :: Ord a => FSet a -> FSet a -> FSet a
member :: Ord a => a -> FSet a -> Bool
```