Kurs języka Haskell

Lista zadań na pracownię nr 1

Na zajęcia 5 i 10 marca 2020

Zadanie 1 (1 pkt). Zaprogramuj samodzielnie poniższe funkcje z modułu `Data.List`. Nie podglądaj ich definicji!

```
intercalate :: [a] -> [[a]] -> [a]
transpose :: [[a]] -> [[a]]
concat :: [[a]] -> [a]
and :: [Bool] -> Bool
all :: (a -> Bool) -> [a] -> Bool
maximum :: [Integer] -> Integer
```

Zadanie 2 (2 pkt). Wektory reprezentujemy w postaci list wartości typu numerycznego:

```
newtype Vector a = Vector { fromVector :: [a] }
```

Zaprogramuj funkcje

```
scaleV :: Num a => a -> Vector a -> Vector a
norm :: Floating a => Vector a -> a
scalarProd :: Num a => Vector a -> Vector a -> a
sumV :: Num a => Vector a -> Vector a -> Vector a
```

Funkcja `scaleV` mnoży wektor przez skalar, `norm` wyznacza długość wektora, `scalarProd` oblicza iloczyn skalarny wektorów, zaś `sumV` dodaje wektory. Jeśli argumenty funkcji są wektorami różnych długości, to obliczenie powinno zakończyć się błędem (użyj funkcji `error`).

Zadanie 3 (2 pkt). Macierze reprezentujemy w postaci list wierszy. Każdy wiersz reprezentujemy w postaci listy wartości typu numerycznego:

```
newtype Matrix a = Matrix { fromMatrix :: [[a]] }
```

Zaprogramuj funkcje

```
sumM :: Num a => Matrix a -> Matrix a -> Matrix a
prodM :: Num a => Matrix a -> Matrix a -> Matrix a
det :: Num a => Matrix a -> a
```

Funkcja `sumM` dodaje, a `prodM` mnoży macierze w tej reprezentacji. Funkcja `det` oblicza wyznacznik macierzy. Jeśli argumenty funkcji są macierzami o nieodpowiednich wymiarach, to obliczenie powinno zakończyć się błędem (użyj funkcji `error`).

Zadanie 4 (1 pkt). Zaprogramuj funkcję o sygnaturze

```
isbn13_check :: String -> Bool
```

która odpowiada na pytanie, czy podany numer ISBN-13 jest poprawny.[1] Przetestuj swoją implementacje na numerach kilku książek, które masz na półce.

[1] Zob. `https://en.wikipedia.org/wiki/International_Standard_Book_Number#ISBN-13_check_digit_calculation`

Zadanie 5 (1 pkt). Dowolnie wielkie liczby całkowite przedstawiamy w Haskellu w postaci list liczb typu `Word` (liczby całkowite bez znaku o rozmiarze słowa maszynowego) w zapisie pozycyjnym przy pewnej podstawie b:

```
newtype Natural = Natural { fromNatural :: [Word] }
```

Lista $[x_0, \ldots, x_{n-1}]$, gdzie $0 \leq x_i < b$ dla $i = 0, \ldots, n-1$ reprezentuje zatem liczbę $\sum_{i=0}^{n-1} x_i b^i$. Ponieważ podczas wykonywania operacji arytmetycznych potrzebujemy mnożyć „cyfry" x_i przez siebie, to największą wartością b gwarantującą, że nie wydarzy się nadmiar stałopozycyjny jest $b = \lfloor\sqrt{\texttt{maxBound}}\rfloor + 1$. Wartość `maxBound` jest jedyną metodą klasy `Bounded`, do której należy typ `Word`. Zdefiniuj stałą

```
base :: Word
```

która ma tę wartość. Liczby typu `Natural` będziemy reprezentować przy tej właśnie podstawie.

Jak dowiemy się dokładniej później, instalacja typu takiego jak `Natural` w pewnej klasie `C` polega na napisaniu deklaracji

```
instance C Natural where
```

i umieszczeniu w zasięgu otwartego bloku `where` definicji wszystkich funkcji, których nazwy są wymienione jako metody w deklaracji klasy `C` (a przynajmniej tych, które są wymienione jako *minimal complete definition*).

W poniższych zadaniach nie korzystaj z funkcji zdefiniowanych dla typu `Integer`. Typ `Natural` ma być w pewnym stopniu jego reimplementacją!

Zadanie 6 (2 pkt). Zainstaluj typ `Natural` z zadania 5 w klasie `Num`.

Zadanie 7 (1 pkt). Zainstaluj typ `Natural` z zadania 5 w klasach `Eq` i `Ord`.

Zadanie 8 (3 pkt). Zainstaluj typ `Natural` z zadania 5 w klasie `Integral`.

Zadanie 9 (1 pkt). Zainstaluj typ `Natural` z zadania 5 w klasie `Show`.

Zadanie 10 (1 pkt). Wyznacz najogólniejsze typy i dopisz sygnatury do poniższych deklaracji. Nie oszukuj i nie korzystaj z kompilatora do wyznaczenia typów!

```
val1  = (.)(.)
val2  = (.)($)
val3  = ($)(.)
val4  = flip flip
val5  = (.)(.)(.)
val6  = (.)($)(.)
val7  = ($)(.)($)
val8  = flip flip flip
val9  = tail $ map tail [[],['a']]
val10 = let x = x in x x
val11 = (\ _ -> 'a') (head [])
val12 = (\ (_,_) -> 'a') (head [])
val13 = map map
val14 = map flip
val15 = flip map
```