# Kurs języka Haskell 2024/25

LISTA NR 7 (TERMIN: 9.12.2024, godz 5:00)

**Uwaga:** Wszystkie rozwiązania należy umieścić w jednym module o nazwie `Lista7` zachowując sygnatury zgodne z szablonem rozwiązań zamieszczonym w SKOS-ie.

**Zadanie 1.** Przypomnij sobie omawianą na wykładzie monadę Writer (linki dostępne także na SKOS-ie):

```
newtype Writer w a = Writer { runWriter :: (w, a) }
```

Zdefiniuj typ Min a taki, że jeśli a jest w klasie **Ord**, to Min a jest monoidem, którego operacja binarna wskazuje minimum. (Każdy typ w klasie **Ord** jest w oczywisty sposób półgrupą, a element neutralny trzeba dodać „sztucznie", np. jak opisano to w sekcji „Adjoining an identity to a semigroup" w https://en.wikipedia.org/wiki/Adjoint_functors#Algebra).

Zdefiniuj funkcję

```
treeMinW :: Tree → Writer (Min Int) ()
```

która oblicza minimalną wartość etykiety w drzewie, korzystając z monady Writer. Użyj tej funkcji jako funkcji pomocniczej w

```
treeMin :: Tree → Int
```

która oblicza minimalną wartość etykiety w drzewie.

**Zadanie 2.** Użyj monady Writer (Min **Int**) do rozwiązania problemu repmin z wykładu nr 3. Odpowiednie funkcje powinny mieć następujące typy:

```
repmin :: Tree → Tree
aux :: Int → Tree → Writer (Min Int) Tree
```

**Zadanie 3.** Rozważ następującą modyfikację monady State, która umie spojrzeć w przyszłość na ostateczny stan obliczenia:

```
newtype StateRes s a =
  StateRes { runStateRes :: s → s → (a, s) }
```

Funkcja wewnątrz konstruktora StateRes ma dwa argumenty: pierwszy to *aktualny* stan, a drugi to *końcowy* stan obliczeń. Oczywiście trzeba uważać: zbyt duża zależność obliczenia od stanu końcowego może skończyć się zapętleniem.

Zainstaluj StateRes w klasach Applicative i **Monad**, a także zdefiniuj następujące funkcje:

```
putr :: s → StateRes s ()   -- zmien aktualny stan
getr :: StateRes s s        -- pobierz aktualny stan
getResult :: StateRes s s   -- pobierz stan koncowy
execStateRes :: StateRes s a → s → (a,s)
    -- uruchom obliczenie ze stanem poczatkowym
```

Funkcja execStateRes powinna zawiązywać węzeł: to ona, uruchamiając obliczenie w monadzie StateRes, karmi je stanem będącym rezultatem obliczenia.

Wykorzystaj monadę execStateRes do zaimplementowania jeszcze jednego rozwiązania problemu repmin: stanem niech będzie minimum wartości przejrzanych do tej pory etykiet (można wykorzystać typ Min z zadania 1.), a każda etykieta zastępowana jest wartością stanu końcowego. Typy mogą więc przedstawiać się następująco:

```
repmin2 :: Tree → Tree
aux2 :: Tree → StateRes (Min Int) Tree
```

**Zadanie 4.** Przypomnij sobie monadę Reader (linki dostępne także na SKOS-ie). Zdefiniuj odpowiadający jej transformator

```
newtype ReaderT s m a =
  ReaderT { runReaderT :: s → m a }
```

Użyj tego transformatora do rozbudowania ewaluatora wyrażeń w odwrotnej notacji polskiej o zmienne, których wartość przechowywana jest w środowisku:

```
type Ident = String
data Cmd = Val Int | Op String | Var Ident
type RPN = [Cmd]
type Env = [(Ident, Int)]
```

Zdefiniuj główną monadę, której użyjesz w implementacji jako synonim typu

```
type RPNMonad a = ...
```

będącego złożeniem Stack, MaybeT oraz ReaderT Env. Jak jest prawidłowa kolejność złożenia transformatorów? Pamiętaj zadbać o to, by brak zmiennej w środowisku powodował błąd (tak jak brak elementów na stosie w wersji z wykładu).

Przykładowe typy funkcji:

```
askEnv :: Ident -> RPNMonad Int
push :: Int -> RPNMonad ()
pop :: RPNMonad Int
evalCmd :: Cmd -> RPNMonad ()
evalRPN :: RPN -> Env -> Maybe Int
```

**Zadanie 5.** Rozważ typ, który mógłby kandydować do roli transformatora odpowiadającego monadzie listowej:

```
newtype ListT m a = ListT { runListT :: m [a] }
```

Niestety, w ogólności nie jest on monadą: dla niektórych monad m odpowiednie równości nie zachodzą. Wybierz taką monadę m i napisz test pokazujący, że prawa dla monad rzeczywiście nie zachodzą.