

Fortgeschrittene Funktionale Programmierung in Haskell

Übungszettel 5

Aufgabe 5.1:

In der Vorlesung wurde `MaybeT` (basierend auf der `Maybe`-Monade) explizit vorgestellt. Erstellen Sie analog einen Monadentransformer `EitherT`, basierend auf der `Either`-Monade.

Zur Erinnerung: `Either` ist definiert als:

```
data Either a b = Left a  -- "Fehler"
                | Right b -- "Erfolg"
```

Erstellen Sie hierzu folgende Instanzen:

- Functor

```
instance Functor f => Functor (EitherT l f) where
    fmap :: (a -> b) -> (EitherT l f) a -> (EitherT l f) b
```

- Applicative

```
instance Applicative f => Applicative (EitherT l f) where
    pure  :: a -> f a
    (<*>) :: Either l f (a -> b) -> Either l f a -> Either l f b
```

- Monad

```
instance Monad m => Monad (EitherT l m) where
    return :: a -> (EitherT l m) a
    (>=>)  :: (EitherT l m) a -> (a -> (EitherT l m) b) -> (EitherT l m) b
```

Beispielcode mit Definitionen und Testfällen finden Sie in der Datei `eitherT.hs`. Nutzen Sie diese als Ausgangsbasis.

Aufgabe 5.2:

Für die folgenden Aufgaben benötigen Sie die externe Bibliothek `mtl`¹, in der der `RWST`-Stack (Read-Write-State-Transformer, siehe Vorlesung) bereits implementiert ist.

Richten Sie sich hierzu mit dem Programm `cabal` in einem Verzeichnis eine lokale Arbeitsumgebung (genannt *sandbox*) ein, indem Sie folgende Befehle verstehen und anschließend ausführen:

```
$ git init                                # git initialisieren - falls gewünscht.
                                           # alternativ: mit "git clone" ein bestehendes
                                           # Repository klonen. Wir helfen gerne dabei.

$ cabal update                            # Paketliste aktualisieren
$ cabal init                              # Erstellen eines Paketes
$ cabal sandbox init                      # Initialisieren der Sandbox
$ nano <projektname>.cabal                # Hinzufuegen von mtl > 2.2.0 && < 2.3
                                           # und transformers >= 4.3.0 && < 4.4
                                           # als Dependency
                                           # Einstellen der Main durch aendern von
                                           #     main-is: game.hs
$ cabal install --only-dependencies        # Installieren aller Dependencies
$ cabal build                             # Projekt bauen
$ cabal run                               # Projekt ausfuehren
$ cabal repl                              # Einen ghci laden, in dem alle Dependencies
                                           # bereits geladen wurden
```

¹<https://hackage.haskell.org/package/mtl>

Aufgabe 5.3:

In dieser Aufgabe geht es um die Verwendung eines **Monad-Stacks**. Hierzu schreiben Sie ein (sehr!) simples Spiel:

Durch drücken von **u** (up) bzw. **d** (down) wird ein interner Counter hoch- bzw. runtergezählt. Arbeiten Sie sich in den gegebenen Code (**game.hs**) ein und erstellen Sie die Game-Loop

```
mainLoop :: RWST Env () State IO ()
```

und die Tasteneingabe

```
getInput :: RWST Env () State IO Input
```

Benutzen Sie hierzu die gegebene *pure* Hilfsfunktion

```
getInputfromEnv :: Char -> Env -> Input
```

Aufgabe 5.4:

Erweitern Sie Ihr Spiel sinnvoll durch einen weiteren Zähler, der durch die Tasten **r** und **l** für rechts und links erhöht bzw. erniedrigt wird.