Aufgaben für die 3. Sitzung

27. August 2017

Umgang mit Stream-, Continuation- oder monadischem IO

Für die folgenden Aufgaben können die Module StreamIO und ContIO von der Website heruntergeladen und eingebunden werden. **Achtung**: Hineingucken verdirbt den Spaß für spätere Aufgaben.

Definiert wird u.A. folgendes:

```
-- StreamIO
type Name = String
data Request = AppendChannel Name String
             | ReadChannel Name
             | AppendFile Name String
             | WriteFile Name String
             | ReadFile Name
             | DeleteFile Name
data Response = Success
              | Return String
              | Failure String
type Behaviour = [Response] -> [Request]

behave :: Behaviour -> IO ()

-- ContIO
type FailCont = String -> Behaviour
type RetCont = String -> Behaviour
type SuccCont = Behaviour

appendChannel, appendFile, writeFile ::
  Name -> String -> FailCont -> SuccCont -> Behaviour
putStr, putStrLn :: String -> FailCont -> SuccCont -> Behaviour
readChannel, readFile :: Name -> FailCont -> RetCont -> Behaviour
deleteFile :: Name -> FailCont -> SuccCont -> Behaviour
exit :: Behaviour
die :: FailCont
```

(1) Schreibe einen kleinen Taschenrechner, der zunächst den Operator einliest (+ und * reichen), dann die beiden Operanden einliest, und dann das Ergebnis ausgibt. Schreibe ihn in drei Varianten:

a) mit Stream-IO (verwende zum Testen behave)

b) mit Continuation-IO (auch hier hilft behave)

c) mit monadischem IO

Tipp: read :: **Read** a ⇒**String** →a

(2) Verwende ein IO-System deiner Wahl um Galgenraten zu implementieren.

(3) Verwende monadisches IO um ein Zahlenraten zu implementieren. Verwende für die Zufallszahl die folgende Funktion aus dem Modul System.Random: (für Int existiert eine Random-Instanz)

```
randomRIO :: Random a ⇒ (a,a) → IO a
```

Umwandlung zwischen IO-Systemen

(4) Implementiere die Funktion

```
behave :: Behaviour → IO ()
```

Dabei können einige Funktionen aus System.IO nützlich sein. Zum Testen kannst du sie auf deine Programme aus (1) anwenden. Außerdem enthält StreamIO die Funktion whatsYourName, die ebenfalls zum Testen nützlich sein kann.

(5) Implementiere die Funktionen

```
writeFile :: Name → String → FailCont → SuccCont → Behaviour
readFile :: Name → FailCont → RetCont → Behaviour
```

ohne auf die Folien zu spicken. Musterlösungen liegen im Modul ContIO.

(6) Rein konzeptuell kann auch monadisches IO in Stream-IO umgewandelt werden. Der Typ aus der Standardbibliothek ist dafür leider nicht geeignet. Das Modul FreeIO von der Kurswebsite enthält jedoch eine alternative Implementierung, mit der das möglich ist. Schau dir den Quelltext an, und vollziehe nach, wie der Typ FreeIO und die Funktion perform :: FreeIO () → Behaviour funktionieren.