



Technische Universität Berlin

Software and Embedded Systems Engineering Group

Prof. Dr. Sabine Glesner

www.sese.tu-berlin.de    Sekr. TEL 12-4    Ernst-Reuter-Platz 7    10587 Berlin







# Softwaretechnik und Programmierparadigmen WiSe 2022/2023

Prof. Dr. Sabine Glesner  
Milko Monecke  
Simon Schwan

## Übungsblatt 02



### Aufgabe 1: Eigene Datentypen

Euer Casino-Rechner beherrscht nun einige Grundfunktionen. Überlegt euch einen geeigneten Datentypen, um Anweisungen zu repräsentieren. Verwendet `deriving Show`, damit Werte der Typen auf der Konsole so ausgegeben werden können wie sie eingegeben werden.

- a) Zunächst soll mit einem Aufzählungstyp `CommandS` nach der gewünschten Berechnung zwischen `Put`, `Take` und `Win` unterschieden werden. Die Berechnungen sollen mit einer Funktion `eval :: CommandS -> Int -> Int -> Int` durchgeführt werden können. 
- b) In einer ersten Erweiterung sollen die einzelnen Anweisungen als Summen- und Produkttyp `CommandP` über ihre Parameter verfügen. Erweitert auch die `eval` Funktion entsprechend. Wie verändert sich der Typ der Funktion `eval`? 
- c) Nun sollen auch noch beliebig komplexe Berechnungen unter Verwendung der drei Funktionen in einem rekursiven Datentyp `CommandR` dargestellt werden, in dem die Operanden jeder Operation entweder wieder Operationen oder Werte `ValR` sein können. Die neue Struktur benötigt eine rekursive `eval` Funktion, die den gesamten Baum auswerten kann.  






---

#### Schlüssel:

-  Ein ergänzendes Video wird zur Vor- oder Nachbereitung veröffentlicht.
-  Wird im Tutorium besprochen.

## Aufgabe 2: Parametrisierte Datentypen





Listen und Tupel sind prominente Vertreter der parametrisierten Datentypen. Schreibt einige Funktionen mit pattern matching für Listen.

- a) Schreibt eine Funktion `rep`, die eine Liste `n`-mal wiederholt. 
- b) Schreibt eine Funktion `mirror`, die eine Liste am Ende spiegelt (z.B., `[1,2]` wird `[1,2,2,1]`). 
- c) Schreibt eine Funktion `drop2`, die die ersten *zwei* Elemente einer Liste entfernt und den Rest zurückgibt. 
- d) In einer Liste vom Typ `CommandR` sind die aktuellen Chips einer Menge von Teilnehmern hinterlegt. Es sollen in der Funktion `kick :: [CommandR] -> [CommandR]` alle Elemente aussortiert werden, die zu 0 evaluieren, also von Teilnehmern die all ihre Chips verspielt haben.
- e) Wieder stellt eine Liste von `CommandR` die “Kontostände” von Besuchern dar. Etwas ist in der letzten Runde schiefgegangen, also sollen alle durch die Funktion `payback :: [CommandR] -> [CommandR]` ihr Geld zurückkriegen, die im letzten Schritt Chips eingesetzt haben. Wenn die letzte Operation ein `TakeR` mit einem `ValR` als zweitem Operand ist soll dies rückgängig gemacht werden. Wenn die Bedingung nicht erfüllt ist, soll nichts verändert werden. 
- f) Manchmal möchte man eine bestimmte Anzahl Chips von einem Pot an mehreren Stapeln verteilen. Das soll in der folgenden Funktion implementiert werden. 





```
share :: CommandR -> [CommandR] -> CommandR -> (CommandR, [CommandR])
```

Es wird von dem Pot (erster Parameter) an jedes Element einer `List CommandR` (zweiter Parameter) einen bestimmten Teil (dritter Parameter) abgegeben werden, d.h. mit `TakeR` vom Pot abgenommen und mit `PutR` zum Element hinzugefügt werden. Ergebnis der Funktion soll der übrige Pot und eine neue Liste mit aktualisierten `CommandR` sein.

### Aufgabe 3: Arbeiten mit Strings

- a) Haskell verwendet Unicode. Gebt alle ASCII-Zeichen (die ersten 7 Bit) mithilfe einer Range aus<sup>1</sup>. 
- b) Einzelne Zeichen können auch im Pattern Matching verwendet werden. Schreibt eine Funktion `isVowel :: Char -> Bool`, die überprüft, ob ein gegebenes Zeichen ein (kleingeschriebener) Vokal ist. 
- c) Erweitert die Funktion zu `hasVowel`, die überprüft, ob ein gegebener String einen Vokal enthält. 
- d) Schreibt eine Funktion `toString`, mit der ein `CommandR` in einen String umgewandelt wird, so dass es kompakter auf der Konsole ausgegeben werden kann. Dabei sollen die Rechenoperationen möglichst “normal” aussehen. 

### Aufgabe 4: Partial application und Komposition

- a) Leitet von dem `^^`-Operator die Funktionen `sqr` ( $n^2$ ) und `pot` ( $2^n$ ) durch partial application ab. 
- b) Leitet aus der `elem`-Funktion<sup>2</sup> die `isVowel`-Funktion ab. 
- c) Haskell bietet die Funktionen `ord :: Char -> Int` und `chr :: Int -> Char` für eine Konvertierung zwischen einem `Char` und ihre numerische r presentation als “Unicode Code Point” an, was f r Buchstaben gleich der ASCII-Kodierung<sup>3</sup> ist. Verwendet Partial Application und Funktionskomposition um aus `ord` und `chr` eine Funktion `lower :: Char -> Char` zu konstruieren, die Gro buchstaben in Kleinbuchstaben  berf hrt. Eingaben, die keine Gro buchstaben sind, m ssen nicht gesondert behandelt werden.  

---

<sup>1</sup> `'A' = '\65'`

<sup>2</sup> entspricht dem mathematischen  $\in$ . `elem :: a -> [a] -> Bool`

<sup>3</sup> z.B.: `ord 'a' = 97`; `ord 'A' = 65`