



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 2023/2024

Prof. Dr. Sabine Glesner

Simon Schwan

Julian Klein

Übungsblatt 01

Organisatorisches

- Anmeldung: über Moses bis zum 30.11.2023 (Abmeldung 10.01.2024 vor Abgabe HA)
- Übungsaufgaben: Eine **unbenotete** Aufgabe, freiwillig aber **empfohlen**
- Hausaufgaben: Eine benotete Hausaufgabe, 30% der Gesamtnote in Einzelarbeit
- Erster Test: 19.01.2024 - 16 Uhr, 20% der Gesamtnote
(Wiederholung 10.04.2024 - 16 Uhr)
- Zweiter Test: 26.02.2024 - 16 Uhr, 50% der Gesamtnote
(Wiederholung 10.04.2024 - 17 Uhr)
- Tutor*innen-Sprechstunden: 2 pro Woche. Termine folgen auf ISIS.
- Organisatorische Fragen an swtpp@sese.tu-berlin.de

Aufgabe 1: Haskell Installation - für Zuhause




Installiert als Vorbereitung für das erste Tutorium Haskell mit Stack. Auf Isis findet ihr in den Literatur- und Softwareempfehlungen eine Anleitung mit GHCup¹ dazu.

¹<https://www.haskell.org/ghcup/>

Schlüssel:

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





Aufgabe 2: Grundlagen Haskell - Stack

- a) Was sind die Unterschiede zwischen *Haskell*, *Cabal* und *Stack*? 
- b) Legt ein neues Haskell Projekt mit *Stack* an (*stack new test-project*). Welche Dateien wurden erzeugt? Probiert folgenden Befehle aus: *stack build*, *stack exec*, *stack test* 
- c) Was ist der Unterschied zwischen den interaktiven Umgebungen *ghci* und *stack repl*? 

Ein besonders gieriges Casino möchte eine Art Taschenrechner für Spielchips entwickeln, mit dem die gängigen Transaktionen inklusive versteckter Gebühren aufgezeichnet und ausgewertet werden können. Besonders perfide: Für Besucher*innen soll es so aussehen als wären das ganz normale Berechnungen. Obwohl das alles wenig durchdacht wirkt und ethisch fragwürdig erscheint, entscheiden wir uns diesen Auftrag anzunehmen, der uns die nächsten Wochen beschäftigen wird.


Aufgabe 3: Funktionen

In diesem Casino wird mit Chips gespielt, die an der Kasse gekauft werden, wo selbstverständlich Transaktionsgebühren anfallen. Alle Chips haben den gleichen Wert. Im ersten Schritt setzen wir die Grundrechenarten als Funktionen mit Integer-Werten um. Es kann davon ausgegangen werden, dass die Integer-Werte nie negativ sind.

- a) Eine konstante Funktion `fee` soll die Transaktionsgebühr festsetzen. Sie soll vorerst einen Chip betragen. 
- b) Die Funktion `charge val` zieht die Gebühr von einem gegebenen Wert ab. Das Ergebnis kann aber nicht negativ werden, d.h. von null Chips wird auch nichts abgezogen. 
- c) Die Funktion `putChips owned added` soll zwei Chip-Anzahlen zusammenrechnen. Sie funktioniert wie die Addition, auf das Ergebnis wird aber die Transaktionsgebühr fällig. 
- d) Die Funktion `takeChips owned taken` zieht einen Chipstapel `taken` von `owned` ab, z.B. wenn man einen Einsatz macht. Da man aber nicht mehr einsetzen kann als man hat, kann das Ergebnis der Operation nicht kleiner als 0 werden - entsprechend einem All-In. 

Aufgabe 4: Rekursion

Euer Casino-Rechner beherrscht nun einige Grundfunktionen, im Casino gibt es aber noch Spezial-Gewinne.

- a) Die Funktion `win :: Int -> Int -> Int` ist ein spezieller Gewinn-Modus für die Spielautomaten, bei dem zwei Chipstapel multipliziert werden. Es gibt aber in den AGBs eine gestaffelte Gebühr, mit der auch verhindert werden soll, dass man einen 

zu großen Gewinn mit einem zu kleinen Einsatz holt. Sei a der größere der beiden Operanden a und b . Dann gibt es a Iterationen, in denen b auf das Ergebnis addiert wird (Multiplikation durch Addition). Jedes Mal wenn die Anzahl der übrigen Iterationen durch 10 teilbar ist, wird b für die nachfolgenden Iterationen um 1 verringert (bis $b=0$ erreicht ist).