



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 01



Aufgabe 1: Organisatorisches

- Anmeldung: über Moses bis zum 30.11.2022 (Abmeldung 15.12.2022 vor Abgabe HA) 
- Übungsaufgaben: Eine **unbenotete** Aufgabe, freiwillig aber **empfohlen**
- Hausaufgaben: Eine benotete Hausaufgabe, 30% der Gesamtnote in Einzelarbeit
- Online-Test: 2.1.2023 - 12 Uhr, 20% der Gesamtnote
(Wiederholung 3.4.2023 - 14 Uhr)
- Präsenzttest: 6.3.2023 - 8 Uhr (oder 9 Uhr), 50% der Gesamtnote
(Wiederholung 30.3.2023 - 17 Uhr)
- Tutor*innen-Sprechstunden: 2 pro Woche. Termine folgen auf ISIS.
- Organisatorische Fragen an swtpp@sese.tu-berlin.de

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*? 





Schlüssel:

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

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). 
- b) Der Casinobetreiber*innen ist der Code nicht hübsch genug, was durch Pattern Matching in der Funktion `win` verbessert werden könnte. Erweitert die bisherige Implementierung entsprechend. 

- c) Entwerft und implementiert eine optimierte Version der `win` Funktion, damit sie weniger Speicher verbraucht. 
- ***)** Zusatz: Um Besucher*innen möglichst lange am Spielen zu halten, sollen am Automaten Einsätze vorgeschlagen werden, die vorangegangene Verluste scheinbar größtenteils wieder ausgleichen können. Die Fibonnacci-Folge hat sich dafür als wirkungsvoll erwiesen. Implementiert eine Funktion `fib :: Int -> Int` und überlegt euch eine mögliche Optimierung. 