universität innsbruck

Funktionale Programmierung | WS 2025/2026 | LVA 703025

Übungsblatt 7, 10 Punkte | Abgabefrist: Mittwoch, 26. November 2025, 6 Uhr

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in Template_07.hs und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.

Aufgabe 1 *Partielle Applikationen* **1 P.**

Wir betrachten die folgenden Funktionen:

```
div1 = (2 /)
div2 = (/ 2)
div3 x = x / 2
div4 x y = x / y
div5 x = (\ y -> y / x)
```

1. Erklären Sie, was `div1` und `div2` tun, und geben Sie für beide den allgemeinsten Typen an. Finden Sie ein Beispiel, das den Unterschied zwischen `div1` und `div2` aufzeigt. (0.5 Punkte)
2. Wir sagen, zwei Haskell Funktionen `f` und `g` sind gleich, wenn `f x1 .. xN = g x1 .. xN` für alle Eingaben `x1`, ..., `xN`. Welche der folgenden Paare von Funktionen sind basierend auf dieser Definition gleich? Begründen Sie ihre Antwort. (0.5 Punkte)
   (a) `div2` und `div3`
   (b) `div2` und `div4 2`
   (c) `div4` und `flip div5`

Aufgabe 2 *Multimengen mit Funktionen höherer Ordnung* **5 P.**

Eine Multimenge ist eine Menge, die Elemente mehrfach enthalten kann. Zum Beispiel hat $M := \{0, 0, 2, 3, 3, 3\}$ 2 Nullen, 1 Zweier, und 3 Dreier, aber keine weiteren Elemente. Alternativ kann man Multimengen auch als Funktionen $A \to \mathbb{N}$ definieren. In diesem Fall entspricht M der Funktion $f_M : \mathbb{N} \to \mathbb{N}$, wobei

$$f_M(x) = \begin{cases} 2 & x = 0 \\ 1 & x = 2 \\ 3 & x = 3 \\ 0 & \text{sonst} \end{cases}.$$

In dieser Aufgabe wollen wir Multimengen mithilfe von Funktionen implementieren. Als Basis dafür dient uns die Datentyp-Definition `data Multiset a = Multiset (a -> Int)` und die Funktion

```
count :: Eq a => a -> Multiset a -> Int
count x (Multiset m) = m x
```

1. Definieren Sie die folgenden Funktionen: (1 Punkt)
   - `empty :: Multiset a` gibt eine leere Multimenge zurück.

- `add :: Eq a => a -> Multiset a -> Multiset a` nimmt ein Element `x` und fügt es einmal zu der gegebenen Multimenge hinzu.
- `remove :: Eq a => a -> Multiset a -> Multiset a` nimmt ein Element `x` und entfernt (falls vorhanden) eine Kopie davon aus der Multimenge.

2. Definieren Sie die folgenden Funktionen: (1 Punkt)
   - `union :: Multiset a -> Multiset a -> Multiset a` erzeugt eine Multimenge, sodass es für jede Kopie eines Elementes in einem der Argumente eine Kopie im Ergebnis gibt.
   - `intersection :: Multiset a -> Multiset a -> Multiset a` erzeugt eine Multimenge, sodass es für jede Kopie eines Elementes im Ergebnis eine Kopie in beiden Argumenten gibt.

3. Definieren Sie die Funktion `image :: (a -> [b]) -> Multiset b -> Multiset a`. Als Argumente nimmt `image` die Umkehrfunktion `f :: a -> [b]` einer Funktion `g :: b -> a` (das heißt `f x` ist die Liste der Elemente `y`, für die `g y == x` gilt) und eine Multimenge `m :: Multiset b`. Sie gibt die Multimenge zurück, die entsteht, wenn man alle Elemente von `m` mit `g` abbildet. Das heißt, für jede Kopie eines Elementes `y :: b` im zweiten Argument, soll es eine Kopie von `g y :: a` im Ergebnis geben. (1 Punkt)

   Hinweise: `sum` berechnet die Summe der Elemente einer Liste.

4. Betrachten Sie die folgenden Funktionsspezifikationen:
   (a) Die Funktion `multisetToList :: Eq a => Multiset a -> [a]` soll eine Liste in die Multimenge bestehend aus den Elementen dieser Liste umwandeln.
   (b) Die Funktion `listToMultiset :: Eq a => [a] -> Multiset a` soll eine Multimenge in eine Liste der Elemente der Multimenge umwandeln.

   Eine dieser beiden Funktionen kann mit unserer Datentyp-Definition nicht implementiert werden. Welche Funktion ist das und warum? Finden Sie einen Typen `T`, sodass die Funktion doch implementiert werden könnte, wenn man `a` durch `T` ersetzt. (2 Punkte)

Aufgabe 3 *Vordefinierte Funktionen höherer Ordnung* **4 P.**

Gegeben ist eine Implementierung eines Systems, welche den Bestand einer Bibliothek als eine Liste aus Abteilungen darstellt. Eine Abteilung besteht aus einem Namen und einer Liste aller zur Abteilung gehörenden Exemplare. Ein Exemplar hat einen Medientyp und einen Titel. Der Zweck dieser Aufgabe ist es, anhand des Beispiels dieses Bibliothekssystems die Anwendung verschiedener vordefinierter Funktionen höherer Ordnung zu üben. Deshalb ist das Definieren rekursiver Funktionen für diese Aufgabe verboten. Versuchen Sie ebenfalls, so wenig wie möglich neue benannte Funktionen zu definieren.
Für diese Aufgabe könnte es hilfreich sein, sich die in der Vorlesung vorgestellten Funktionen `map`, `filter` und `elem` noch einmal anzusehen.

1. Implementieren Sie eine Funktion `largestSection :: Library -> String`, welche eine Bibliothek als Eingabewert erhält und den Namen der Abteilung davon, welche die meisten Exemplare beinhaltet, zurückgibt. Die Funktion `maximum` könnte sich hierbei als hilfreich erweisen. (1 Punkt)
   Beispiele:
   - ```
     largestSection [
             Section "Fiction" [Item "Twilight" Book, Item "A Song of Ice and Fire" Book],
             Section "Textbooks" [Item "Real Wolrd Haskell" Book]
         ] = "Fiction"
     ```

2. Implementieren Sie die Funktion `searchItem :: Section -> String -> [Item]`. Die Funktion erhält einen String sowie eine Abteiltung und gibt alle Exemplare dieser Abteilung, deren Titel den gegeben String enthalten, zurück. Hierbei soll Groß- und Kleinschreibung egal sein. Sie können für diese Aufgabe die Funktionen `toLower` und `isInfixOf` verwenden. (1 Punkt)
   Beispiele:
   - ```
     searchItem (Section "Fiction" [
             Item "Twilight" Book,
             Item "A Song of Ice and Fire" Book
         ]) "fire" = [Item "A Song of Ice and Fire" Book]
     ```

- ```
  searchItem (Section "Fiction" [
          Item "Twilight" Book,
          Item "A Song of Ice and Fire" Book
      ]) "I" = [Item "Twilight" Book, Item "A Song of Ice and Fire" Book]
  ```
- ```
  searchItem (Section "Fiction" [
          Item "Twilight" Book,
          Item "A Song of Ice and Fire" Book
      ]) "Interview" = []
  ```

3. Implementieren Sie die Funktion `singleType :: Section -> Bool`, welche für die gegebene Abteilung ermittelt, ob alle Exemplare darin dasselbe Medium haben. Die Funktion `all` könnte sich hierbei als hilfreich erweisen. (1 Punkt)
   Beispiele:
   - ```
     singleType (Section "Fiction" [
         Item "Twilight" Book,
         Item "A Song of Ice and Fire" Book
     ]) = True
     ```
   - ```
     singleType (Section "Fiction" [
         Item "Twilight" Book,
         Item "Barbie" DVD
     ]) = False
     ```

4. Implementieren Sie eine Funktion `filterByMedium :: Library -> [Medium] -> Library`. Hierbei erhält `filterByMedium` eine Bibliothek und eine Liste aus Medientypen als Eingabe. Zurückgegeben wird eine Version der Bibliothek, die nur noch Exemplare enthält, welche einen der gegebenen Medientypen haben. Abteilungen, welche keine Exemplare des gegebenen Typen beinhalten, sollen nicht in in der zurückgegebenen Bibliothek beinhaltet sein. (1 Punkt)
   Beispiele:
   - ```
     filterByMedium [
         Section "Fiction" [Item "Twilight" Book, Item "Barbie" DVD],
         Section "Textbooks" [Item "Real Wolrd Haskell" Book]
     ] [Book, BookChapter] = [
         Section "Fiction" [Item "Twilight" Book],
         Section "Textbooks" [Item "Real Wolrd Haskell" Book]
     ]
     ```
   - ```
     filterByMedium [
         Section "Fiction" [Item "Twilight" Book, Item "Barbie" DVD],
         Section "Textbooks" [Item "Real Wolrd Haskell" Book]
     ] [DVD] = [Section "Fiction" [Item "Barbie" DVD]]
     ```