universität innsbruck

Funktionale Programmierung	WS 2026/2027	LVA 703206/703025
Übungsblatt 3, 10 Punkte	Abgabefrist: Dienstag, 10. November 2026, 20:00 Uhr	

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in Template_03.hs und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.

Aufgabe 1 *Pattern Matching* **2 P.**

Betrachten Sie die folgenden Datentypdefinitionen: (2 Punkte)

```
data Course
  = FunctionalProgramming -- Funktionale Programmierung
  | LinearAlgebra         -- Lineare Algebra
  | Algorithms            -- Algorithmen und Datenstrukturen
  | OperatingSystems      -- Betriebssysteme

data Registration
  = Lecture Course        -- VO
  | Exercise Course       -- UE
  | Tutor Course Course   -- Tutor:innen betreuen zwei Lehrveranstaltungen

data Student
  = Student
      String              -- Name
      Integer             -- Matrikelnummer
      Bool                -- aktiv inskribiert?
      Registration
```

Gegeben sind vier Ausdrücke (1.–4.) und zwei Muster ((i) und (ii)). Bestimmen Sie für jedes Muster, welche der vier Ausdrücke darauf passen.
Geben Sie für jede Übereinstimmung die resultierende Substitution an, d. h. den Wert, der an jede Variable des Musters gebunden wird.

Ausdrücke:

1. `Student "Maxime Majora" 123456 True (Lecture OperatingSystems)`
2. `Student "Warner Wario" 666666 False (Lecture FunctionalProgramming)`
3. `Student "Gan Ganondorf" 654321 False (Tutor LinearAlgebra Algorithms)`
4. `Student "Bowser Jr." 111111 False (Tutor Algorithms LinearAlgebra)`

Muster:

(i) `Student name _ _ (Tutor _ LinearAlgebra)`

(ii) `Student name n False registration@(Lecture _)`

Aufgabe 2 *Funktions-Definitionen* **3 P.**

Wir betrachten die folgenden Datentypen:

```
data Date = DMY Int Int Integer deriving Show
data List = Cons Integer List | Empty deriving Show
```

Hinweis: Ein Wert `DMY d m y` steht für den Tag `d`, den Monat `m` und das Jahr `y`. Die folgenden Funktionen können in Haskell Zahlen miteinander vergleichen: `==`, `<`, `>`, `<=`, `>=`.

1. Definieren Sie eine Funktion `probablyValidDate :: Date -> Bool`, welche überprüft, ob ein Datum die folgenden Bedingungen erfüllt: (1 Punkt)
   - Der Tag ist zwischen 1 und 31 (jeweils einschließlich)
   - Der Monat ist zwischen 1 und 12 (jeweils einschließlich)
2. Definieren Sie eine Funktion `maxList :: List -> Integer`, welche eine Liste nimmt, und ihr größtes Element zurückgibt. Bei einer leeren Liste soll eine Fehlermeldung mit dem Text „maxList on empty list“ geworfen werden. (1 Punkt)

   Hinweis: Die Funktion `max` gibt das größere von zwei Elementen zurück.
3. Definieren Sie eine Funktion `addSecond :: Integer -> List -> List` welche eine Zahl und eine Liste nimmt, und diese Zahl zu jedem zweiten Element der Liste addiert (also zum 2., 4., 6., ... Element). Wenn die Funktion zum Beispiel mit 2 und der Repräsentation von $[2, 7, 9, 3, 15, 5, 7]$ aufgerufen wird, sollte sie die Repräsentation von $[2, 9, 9, 5, 15, 7, 7]$ zurückgeben. (1 Punkt)

Aufgabe 3 *Rekursive Datentypen und Funktionen* **5 P.**

Betrachten Sie den folgenden Datentypen:

```
data QuadTree
  = Black
  | White
  | Node QuadTree QuadTree QuadTree QuadTree
```

Dieser Datentyp repräsentiert einen Schwarz-Weiß-Quadtree[1] der Anwendung in der Bildverarbeitung findet.

1. Implementieren Sie eine rekursive Funktion `blackRatio :: QuadTree -> Double`, die das relative Verhältnis der schwarzen Fläche zur Gesamtfläche berechnet, so dass das Ergebnis für `Black` stets `1.0` und für `White` stets `0.0` ergibt. Für einen `Node` soll das Ergebnis dem Mittelwert der vier Quadranten entsprechen. (1 Punkt)

   Beispiel: `blackRatio (Node Black White White Black) = 0.5`
2. Implementieren Sie eine rekursive Funktion `compress :: QuadTree -> QuadTree`, die redundante Teilbäume zusammenfasst, so dass vier identische Quadranten eines `Node`s (alle `Black` oder alle `White`) durch ein einzelnes Blatt ersetzt werden. (2 Punkte)

   Beispiel: `compress (Node Black Black Black Black) = Black`
3. Implementieren Sie eine Funktion `overlay :: QuadTree -> QuadTree -> QuadTree`, die zwei `QuadTree`s logisch überlagert (ODER-Verknüpfung), wobei `Black` stets `White` überdeckt. (2 Punkte)

   Beispiel: `overlay Black (Node White Black White White) = Black`

[1] https://en.wikipedia.org/wiki/Quadtree