universität innsbruck

Funktionale Programmierung	WS 2025/2026	LVA 703025
Übungsblatt 8, 10 Punkte		Abgabefrist: Mittwoch, 03. Dezember 2025, 6 Uhr

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in Template_08.hs und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.

Aufgabe 1 *List Comprehension* **5 P.**

Für die folgenden Aufgaben darf **keine** Rekursion verwendet werden, sondern ausschließlich **List Comprehension**.

1. Erstellen Sie die Funktion `divisors :: Int -> [Int]`, welche eine positive ganze Zahl als Argument nimmt und die Liste aller Teiler zurückgibt (außer die Zahl selbst). (0.5 Punkte)

   Beispiel: `divisors 28 = [1,2,4,7,14]`

2. Eine Zahl wird als perfekt bezeichnet, wenn sie gleich der Summe all ihrer Teiler ist. Implementieren Sie die Funktion `isPerfect :: Int -> Bool`, welche prüft ob eine Zahl perfekt ist. Anschließend sollen Sie die Funktion `perfectNumberUpTo :: Int -> [Int]` implementieren, welche eine positive ganze Zahl n nimmt und als Ergebnis die Liste aller perfekten Zahlen bis zu n zurückgibt. (0.5 Punkte)

   Beispiel: `perfectNumberUpTo 28 = [6,28]`

3. Schreiben Sie die Funktion `pairNumbers :: Int -> [(Int,Int)]`, welche als Eingabe eine positive Zahl n nimmt und als Ergebnis eine Liste geordneter Paare der Form (a, b) liefert, wobei $1 \leq a < b \leq n$, b gleich der Summe der echten Teiler von a ist und a gleich der Summe der echten Teiler von b ist. (1 Punkt)

   Beispiel: `pairNumbers 300 = [(220,284)]`

4. Erstellen Sie die Funktion `removeNonLetters :: String -> String`, welche alle Zeichen eines Strings entfernt, die keine Buchstaben sind oder Leerzeichen. (1 Punkt)

   Beispiel: `removeNonLetters "Da2s i1s,t ein_ Test" = "Das ist ein Test"`

5. Gegeben sind die importierten Funktionen `ord` und `chr`, welche nützlich sind, um einen `Char` in ein `Int` umzuwandeln und umgekehrt. Nutzen Sie diese Funktionen, um die Funktion `shiftRight :: Int -> String -> String` zu erstellen, welche alle Kleinbuchstaben eines Strings um n Stellen nach rechts verschiebt. Ist n zum Beispiel 3, so wird aus einem `c` ein `f` und aus einem `x` ein `a`. (1 Punkt)

6. Zum Schluss sollen Sie die Funktion `everyNth :: Int -> String -> String` erstellen, welche aus einem String nur jeden n-ten Buchstaben auswählt. (1 Punkt)

   Beispiel: `everyNth 2 "abcdefg" = "bdf"`
   Wenn Sie die Funktionen `removeNonLetters`, `shiftRight` und `everyNth` korrekt implementiert haben, können Sie diese in der Funktion `crackingTheCode :: String -> [String]` kombinieren, um ein gegebenes Codewort zu entschlüsseln. Dabei sollen im Text zuerst alle Zeichen entfernt werden, die keine Buchstaben oder Leerzeichen sind, dann jeder n-te Buchstabe extrahiert werden und schließlich jeder Kleinbuchstabe um m Stellen verschoben werden. Um die richtigen Werte für n und m herauszufinden, kann mit einer List Comprehension experimentiert werden. Dabei ist $m \leq 7$ und $n \leq 5$.

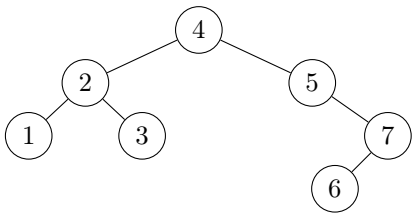
Aufgabe 2 *Fold Functions* **5 P.**

Gegeben ist der folgende Datentyp für einen binären Baum.

```
data Tree a = Leaf | Node (Tree a) a (Tree a)

tree1, tree2 :: Tree Int
tree1 = Node (Node Leaf 1 Leaf) 2 (Node Leaf 3 Leaf)
tree2 = Node tree1 4 (Node Leaf 5 (Node (Node Leaf 6 Leaf) 7 Leaf))
```

Der Baum `tree2` kann wie folgt dargestellt werden:



(i) Implementieren Sie die Funktion `foldT :: (b -> a -> b -> b) -> b -> Tree a -> b`, wobei die Funktion `foldT f e t` jeden Knoten `Node` in dem Baum `t` mit `f` und jedes Blatt `Leaf` in dem Baum `t` mit `e` ersetzt.

Beispiel: `foldT (\accL x accR -> accL + x + accR) 0 tree2 == 28` (1 Punkt)

(ii) Implementieren Sie folgende Funktionen mit Hilfe von `foldT`: (1 Punkt)

- `height` berechnet die Höhe des Baums.

  Beispiel: `height tree2 == 4`
- `size` berechnet die Größe (Anzahl der Knoten) des Baums.

  Beispiel: `size tree2 == 7`
- `sumT` berechnet die Summe der Knoten des Baums.

  Beispiel: `sumT tree2 == 28`

(iii) Implementieren Sie die folgenden Funktionen auf Bäumen mit Hilfe von `foldT`: (1 Punkt)

- `mirror` soll den linken und rechten Teilbaum jedes Knotens vertauschen.

  Beispiel: `mirror tree1 == Node (Node Leaf 3 Leaf) 2 (Node Leaf 1 Leaf)`
- `mapT` soll eine Funktion auf jeden Knoten im Baum anwenden, wobei die Baumstruktur erhalten bleibt.

  Beispiel: `mapT (*2) tree1 == Node (Node Leaf 2 Leaf) 4 (Node Leaf 6 Leaf)`

(iv) Implementieren Sie die folgenden Traversierungsfunktionen mit Hilfe von `foldr` und `foldT`: (1 Punkt)

- `inOrder` soll die Werte eines Baums in der Reihenfolge *left* → *root* → *right* ausgeben.

  Beispiel: `inOrder tree2 == [1,2,3,4,5,6,7]`
- `preOrder` soll die Werte eines Baums in der Reihenfolge *root* → *left* → *right* ausgeben.

  Beispiel: `inOrder tree2 == [4,2,1,3,5,7,6]`
- `postOrder` soll die Werte eines Baums in der Reihenfolge *left* → *right* → *root* ausgeben.

  Beispiel: `postOrder tree2 == [1,3,2,6,7,5,4]`
- `fromList` soll aus einer gegebenen Liste einen Baum erzeugen, wobei `inOrder (fromList xs) == xs`.

(v) Schauen Sie sich die Data.Foldable Klasse an. Implementieren Sie eine Instanz für `Foldable Tree`, sodass `toList t == inOrder t` für jeden Baum `t` gilt. (1 Punkt)