universität innsbruck

Funktionale Programmierung	WS 2025/2026	LVA 703025
Übungsblatt 11, 10 Punkte		Abgabefrist: Mittwoch, 21. Januar 2026, 6 Uhr

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in `Template_11.hs` und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.

Aufgabe 1 *Auswertungs-Strategien* **5 P.**

Betrachten Sie die folgenden Funktionen.

```
-- program 1
append [] ys = ys
append (x : xs) ys = x : append xs ys

filter f [] = []
filter f (x : xs)
  | f x = x : filter f xs
  | otherwise = filter f xs

reverse [] = []
reverse (x : xs) = append (reverse xs) [x]

-- program 2
addTwice x y = x + (y + y)

take 0 _ = []
take _ [] = []
take n (x : xs) = x : take (n - 1) xs

zipWith f [] _ = []
zipWith f _ [] = []
zipWith f (x : xs) (y : ys) = f x y : zipWith f xs ys
```

1. Werten Sie den Ausdruck `filter (== 2) (reverse [3, 2, 1])` mit den folgenden zwei Strategien Schritt für Schritt aus (siehe Folie 11/8).
   - (a) innermost (1 Punkt) und (b) outermost (1 Punkt)
2. Werten Sie den Ausdruck `take 1 (zipWith addTwice [1 + 2, 3] [4 + 5, 6])` mit den folgenden drei Strategien Schritt für Schritt aus.
   - (a) innermost (1 Punkt), (b) outermost (1 Punkt), und (c) lazy evaluation (1 Punkt)

Aufgabe 2 *Unendliche Listen, Rekursion* **5 P.**

1. Implementieren Sie eine rekursive Funktion `applyIndefinitely`, die zu einer gegebenen Funktion `f` und einem Anfangswert `x` die unendliche Liste der wiederholten Anwendungen von `f` auf `x` erzeugt, also

   ```
   [x, f x, f (f x), f (f (f x)), ...]
   ```

und verwenden Sie diese Funktion, um die unendliche Liste `infDigits` der Ziffern

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, .., 9, 0, 1, .., 9, 0, ...]
```

zu erzeugen. (1 Punkt)

2. Betrachten Sie das folgende Haskell-Programm, das Funktionen zur Erzeugung einer Menge von Elementen aus einer Liste von Elementen bereitstellt.

```
import OList_09

fromListA :: Ord a => [a] -> OList a
fromListA [] = empty
fromListA (x : xs) = insert x (fromListA xs)

fromListB :: Ord a => [a] -> OList a
fromListB = flb empty

flb s [] = s
flb s (x : xs) = flb (insert x s) xs
```

Klassifizieren Sie die Art der Rekursion für `fromListA` und `flb`. Sind diese Funktionen definiert durch

- verschränkte Rekursion?
- geschachtelte Rekursion?
- lineare Rekursion?
- guarded recursion?
- Endrekursion? (1 Punkt)

3. Betrachten Sie eine Erweiterung des vorherigen Programms.

```
numberInsertions = 2 * 10^6
testList = take numberInsertions infDigits

testA, testB :: String
testA = show $ fromListA testList
testB = show $ fromListB testList
```

   (a) Schreiben Sie eine Funktion `testWithArgs`, die entweder `testA` oder `testB` (ausgehend von einem Kommandozeilenargument) auswertet und ausgibt. Führen Sie das Programm wie folgt aus: `cabal run Main -- A +RTS -s -RTS` um das Ergebnis von `testA` ausgeben, wobei die ghc-Parameter `+RTS -s -RTS` zusätzliche Ausgaben in Form von Auswertungsstatistiken erzeugen. (1 Punkt)

   (b) Suchen Sie in den Statistiken nach dem Speicherverbrauch von `testA` und `testB`, der durch eine Zeile „total memory in use“ angegeben ist. Geben Sie die Werte an und erklären Sie, warum diese so hoch sind, obwohl die resultierende Menge nur 10 Elemente speichert. (1 Punkt)

   (c) Schreiben Sie eine Funktion `fromListC`, sodass der Speicherverbrauch gering bleibt, d.h. unter 10 MiB liegt. (1 Punkt)