WS 2017/2018

Funktionale Programmierung

7. Übungsblatt

Prof. Dr. Margarita Esponda

(Abgabetermin: Mi., den 13.12., um 10:10, Hörsaal)

Ziel: Auseinandersetzung mit algebraischen Datentypen und Funktionen höherer Ordnung.

1. Aufgabe (11 Punkte)

a) (8 P.) Programmieren Sie folgende Funktionen für den algebraischen Datentyp **ZInt** aus der Vorlesung.

```
maxZ  ::  ZInt -> ZInt -> ZInt  -- berechnet die größte Zahl
absZ  ::  ZInt -> ZInt          -- absoluter Wert einer Zahl
isTeilerZ :: ZInt -> ZInt -> B  -- überprüft, ob die zweite Zahl Teiler der ersten Zahl ist.
ggtZ  ::  ZInt -> ZInt -> ZInt  -- größter gemeinsamer Teiler
```

b) (2 P.) Definieren Sie die Hilfsfunktionen **zint2Int** und **int2ZInt**, die das Testen der Funktionen für **ZInt** vereinfachen sollten. Ausnahmsweise ist hierfür erlaubt vordefinierte Haskell Funktionen zu verwenden.

Anwendungsbeispiel:

```
zint2Int  (Z (S (S (S Zero))) Zero)   =>   -3
int2ZInt  (-5)   =>  Z 5 0    — mit eigener Instanz/show-Funktion von Nat in Show
int2ZInt  (-5)   =>  Z (S (S (S (S (S Zero))))) Zero — mit "deriving Show" in Nat
```

c) (1 P.) Deklarieren Sie Ihren **ZInt** Typ als Instanz der **Show** Klasse, indem Sie eine eigene **show** Funktion dafür definieren.

2. Aufgabe (8 Punkte)

Betrachten Sie folgende algebraische Datentyp-Definition (Baumstruktur ohne Inhalt aus der Vorlesung).

```
data SimpleBT = L | N SimpleBT SimpleBT   deriving Show
```

Definieren Sie folgende Funktionen damit:

```
insertLeaves :: Integer -> SimpleBT -> SimpleBT
                -- eine eingegebene Anzahl von Blättern wird eingefügt.

deleteLeaves :: Integer -> SimpleBT -> SimpleBT
                -- eine eingegebene Anzahl von Blättern wird gelöscht.
```

Alle Funktionen sollen selber dafür sorgen, dass der Baum möglichst balanciert bleibt.

3. Aufgabe (14 Punkte)

Erweitern Sie die Funktionen für den algebraischen Datentyp **BSearchTree** (Binäre Suchbäume) um folgende Funktionen:

```
postOrder :: (Ord a) => BSearchTree a -> [a]
```

-- siehe die Definition in den Vorlesungsfolien

twoChildren :: (Ord a) => BSearchTree a -> Bool

-- entscheidet, ob jeder Knoten des Baums genau zwei Kinder hat oder nicht

full :: (Ord a) => BSearchTree a -> Bool

-- überprüft, ob ein Baum vollständig ist.

mapTree :: (Ord a, Ord b) => (a -> b) -> BSearchTree a -> BSearchTree b

foldTree :: (Ord a) => b -> (a -> b -> b -> b) -> BSearchTree a -> b

successor :: (Ord a) => a -> BSearchTree a -> Maybe a

-- berechnet den Nachfolger des eingegebenen Objektes

4. Freiwillige Aufgabe (6 Bonuspunkte)

a. Definieren Sie einen abstrakten Datentyp **Queue**, indem Sie eine Warteschlange mit den Operationen **enqueue** (Einfügen) , **dequeue** (Entfernen), **isEmpty** und **makeQueue** (Erzeugt eine leere Warteschlange) modellieren.

b. Definieren Sie eine geeignete Funktion **showQueue**, die als Hilfsfunktion verwendet werden soll, um den **Queue**-Datentyp als Instanz der **Show**-Typklasse zu deklarieren.

c. Definieren Sie geeignete Gleichheit und Vergleich-Infix-Operatoren für Ihren Queue Datentyp und deklarieren Sie damit den **Queue**-Datentyp als Instanz der **Eq**- und **Ord**-Typklassen.

d. Schreiben Sie geeignete Testfunktionen für alle Funktionen Ihres **Queue**-Datentyps.

Damit die **dequeue** Operation auf die Verwendung der (++) Funktion verzichten kann, modellieren Sie Ihre Warteschlange mit Hilfe von zwei Listen. Elemente werden immer aus der ersten Liste entfernt und neue Elemente werden am Anfang der zweiten Liste eingefügt. Wenn die erste Liste leer ist und ein weiteres Element entfernt werden soll, wird die zweite Liste umgedreht und als erste Liste gesetzt.

5. Freiwillige Aufgabe (6 Bonuspunkte)

Definieren Sie einen abstrakten Datentyp **ABaum**, der, anders als die Binärbäume, nicht nur zwei Kinder, sondern beliebig viele Kinder haben darf. Die Kinder können in Listen gespeichert werden.

Schreiben Sie für Ihren **ABaum**-Datentyp folgende Funktionen:

nodes -- zählt die gesamte Anzahl der inneren Knoten.

height -- berechnet die Höhe eines Baumes.

mapTree -- eine map-Funktion für den Baum.

Wichtige Hinweise:

1. Schreiben Sie in alle Funktionen die entsprechende Signatur.
2. Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.
3. Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.
4. Kommentieren Sie Ihre Programme.
5. Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.