6. Aufgabenblatt zu Funktionale Programmierung vom 13.11.2007. Fällig: 20.11.2007 / 27.11.2007 (jeweils 15:00 Uhr)

Themen: *Typklassen, Polymorphie und Überladung*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe6.lhs` ablegen. Wie für die Lösung zum zweiten Aufgabenblatt sollen Sie dieses Mal also wieder ein "literate" Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

Wir betrachten den folgenden Datentyp für Bäume:

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

Wir sagen, dass ein Baum s *Strukturanfangsbaum* eines Baums t ist, wenn s aus t dadurch entsteht, dass einige Vorkommen von inneren Knoten in t (kenntlich durch den Konstruktor `Node`) samt der darunter hängenden Teilbäume durch Blätter (kenntlich durch den Konstruktor `Leaf`) ersetzt sind und zusätzlich Benennungen von Blättern und inneren Knoten in t durch möglicherweise andere Benennungen in s ersetzt sind.

1. Machen Sie den Typ `Tree a` mithilfe expliziter Instanzdeklarationen `instance Eq a => ...` und `instance Ord a => ...` zu Instanzen der Typklassen `Eq` und `Ord`.

   Geben Sie dazu Implementierungen der von den Typklassen `Eq` und `Ord` erforderten Funktionen an, so dass gilt:

   - Zwei Bäume sind genau dann *gleich*, wenn sie in Struktur und Benennungen übereinstimmen.
   - Ein Baum s ist *echt kleiner* als ein Baum t genau dann, wenn s ein Strukturanfangsbaum von t ist und alle Benennungen von s echt kleiner sind als die entsprechenden Benennungen in t.
   - Ein Baum s ist *kleiner oder gleich* als ein Baum t genau dann, wenn s ein Strukturanfangsbaum von t ist und alle Benennungen in s kleiner oder gleich sind als die entsprechenden Benennungen in t.

   Für die Implementierung der Funktionen `max`, `min` und `compare` wird nichts gefordert. Nutzen Sie, wo es möglich ist, die default-Implementierungen der Klassenfunktionen aus.

2. Schreiben Sie Haskell-Rechenvorschriften

   ```
   lsortIncr :: (Ord a) => [a] -> [a]
   lsortDecr :: (Ord a) => [a] -> [a]
   tsortIncr :: (Ord a) => (Tree a) -> (Tree a)
   tsortDecr :: (Ord a) => (Tree a) -> (Tree a)
   ```

   Dabei soll gelten, dass auf `sortIncr` endende Funkionen ihr Argument aufsteigend sortieren, auf `sortDecr` endende Funktionen absteigend.

   Auf Bäumen soll gelten, dass die Funktion `tsortIncr` (`tsortDecr`) die Knoten im Baum so umordnet (d.h. einen neuen Baum mit umgeordneten Knoten aufbaut und zurückgibt), dass die Benennungen in aufsteigender (absteigender) Reihenfolge ausgegeben werden, wenn der Resultatbaum in Infixordnung durchlaufen wird. Die Resultatbäume sollen dabei möglichst gut balanciert sein; d.h. die Tiefe eines linken Teilbaums soll sich von der eines rechten Teilbaums höchstens um 1 unterscheiden.

   Auf Listen soll gelten, dass die Funktion `lsortIncr` (`lsortDecr`) als Resultat eine Liste zurückliefert, die die Elemente der Argumentliste in aufsteigender (absteigender) Folge enthält.

3. Schreiben Sie Haskell-Rechenvorschriften

```
ulsortIncr :: (Ord a) => [a] -> [a]
ulsortDecr :: (Ord a) => [a] -> [a]
utsortIncr :: (Ord a) => (Tree a) -> (Tree a)
utsortDecr :: (Ord a) => (Tree a) -> (Tree a)
```

   Für die Funktionen `u*sortIncr` (`u*sortDecr`) gelten die gleichen Anforderungen wie für ihre Gegenstücke aus der vorigen Teilaufgabe, jedoch sollen die Resultate für Listen gänzlich frei von Duplikaten sein, für Bäume darf höchstens die kleinste vorkommende Benennung zweimal im Resultatbaum enthalten sein, um eine stets ungerade Anzahl von Benennungen sicherzustellen.

Zum Überlegen (*ohne Abgabe*): Ist es möglich eine Typklasse `Sortable` mit zwei Funktionen `sortIncr` und `sortDecr` und eine Typklasse `UniqueSortable` mit zwei Funktionen `usortIncr` und `usortDecr` so zu definieren, dass man Listen vom polymorphen Typ `[b]` und Bäume vom polymorphen Typ `Tree b` über geordneten Typen `b` (d.h. "`b` aus `Ord`") zu Instanzen der Typklassen `Sortable` und `UniqueSortable` machen könnte?

Denken Sie bitte daran, dass für die Lösung von Aufgabenblatt 6 ein "literate" Haskell-Skript gefordert ist!