Programmierung

02. Übungsblatt

Zeitraum: 27. – 30. April 2015

Übungsverlegungen wegen 01. Mai:

- *Ü Fr., 1. DS, WIL/C204 findet statt am 30.04., APB/3027, 6.DS.*
- *Ü Fr., 4. DS, WIL/C103 findet statt am 29.04., APB/E08, 3.DS.*
- *Ü Fr., 4. DS, WIL/C104 findet statt am 29.04., APB/E06, 4.DS.*

Übung 1 (AGS 12.1.21 ⋆)
(a) Geben Sie in Haskell eine boolesche Funktion `compare` einschließlich der Typdefinition an, die zwei Listen mit Elementen des Typs `Int` auf Gleichheit prüft.

(b) Schreiben Sie in Haskell eine Funktion `merge` einschließlich der Typdefinition, die aus zwei aufsteigend geordneten Listen mit Elementen des Typs `Int` durch Zusammenfügen eine aufsteigend geordnete Liste erzeugt.

Übung 2
(a) Schreiben Sie eine Funktion `pack :: [Char] -> [[Char]]`, welche in einer Liste aufeinander folgende Wiederholungen des gleichen Werts in einer Teilliste zusammenfasst.
Z.B.: `pack ['a','a','b','b','b','a'] = [['a','a'], ['b','b','b'], ['a']]`.

(b) Schreiben Sie eine Funktion `encode :: [Char] -> [(Int, Char)]`, welche eine Liste Lauflängenkodiert.
Z.B.: `encode ['a','a','b','b','b','a'] = [(2, 'a'), (3, 'b'), (1, 'a')]`.

(c) Schreiben Sie eine Funktion `decode :: [(Int, Char)] -> [Char]`, welche eine Lauflängenkodierte Liste wieder dekodiert.
Z.B.: `decode [(2, 'a'), (3, 'b'), (1, 'a')] = ['a','a','b','b','b','a']`.

(d) Schreiben Sie eine Funktion `rotate :: [Int] -> Int -> [Int]`, so dass `rotate xs n` die Liste `xs` um `n` nach links rotiert.
Z.B: `rotate [1,2,3,4] 1 = [2,3,4,1]` oder `rotate [1,2,3] (-1) = [3,1,2]`.

Übung 3 (AGS 12.1.9)
Gegeben sei die folgende Typvereinbarung für binäre Bäume:

```
data Tree = Leaf Int | Branch Tree Tree
```

(a) Schreiben Sie einen Baum dieses Typs mit mindestens 5 Blättern auf.
(b) Schreiben Sie folgende Funktionen für o.g. Datentyp auf:
1. Zum Ermitteln der Anzahl der Blätter.
2. Zum Erstellen einer Liste aller Blattinformationen (von links nach rechts gelesen).

Übung 4 (AGS 12.1.39 ⋆)
Aus der Vorlesung ist bereits die Darstellung von arithmetischen Ausdrücken mittels algebraischer Datentypen bekannt. Wir wollen diese nun um Variablen und um Exponentiation erweitern, d.h. wir betrachten den folgenden Datentyp.

```
data Expr = Lit Int
          | Var Char
          | Add Expr Expr
          | Mul Expr Expr
          | Exp Expr Int
type Assignment = Char -> Int
```

(a) Erweitern Sie die Auswertungsfunktion `eval` aus der Vorlesung um die Auswertung von Variablenbelegungen, d.h. zu einer Funktion `eval :: Expr -> Assignment -> Int`. Dabei soll eine Variable `Var 'x'` zu ihrem Wert unter der übergebenen Variablenbelegung evaluiert werden.

(b) Schreiben Sie eine Funktion `display :: Expr -> String`, welche den übergebenen Ausdruck als String darstellt!

(c) *(Zusatzaufgabe)* Schreiben Sie nun eine Funktion `diff :: Expr -> Char -> Expr`, so dass `diff e 'x'` eine (symbolische) Ableitung des Ausdrucks `e` nach `x` ist. So könnte der Aufruf von `diff e 'x'` mit

```
e = Add (Mul (Lit 5) (Exp (Var 'x') 2))
        (Mul (Var 'x') (Var 'y'))
```

beispielsweise den Wert

```
Add (Add (Mul (Lit 0) (Exp (Var 'x') 2))
         (Mul (Lit 5) (Mul (Mul (Lit 2) (Exp (Var 'x') 1))
                           (Lit 1))))
    (Add (Mul (Lit 1) (Var 'y')) (Mul (Var 'x') (Lit 0)))
```

ergeben.

Zusatzaufgabe 1 (AGS 12.1.12)
Gegeben sei der folgende Datentyp `Tree`.

```
data Tree = Node Int Tree Tree | Nil
```

(a) Definieren Sie eine Haskell-Funktion `check :: Tree -> Bool`, die überprüft, ob es sich bei einem Baum vom Typ `Tree` um einen binären Suchbaum handelt, das heißt jeder Knoten ist mit einer ganzen Zahl beschriftet und es muss für jeden Knoten `x` gelten, dass seine Beschriftung größer oder gleich (bzw. kleiner oder gleich) allen Beschriftungen im linken (bzw. rechten) Teilbaum von `x` ist.

(b) Definieren Sie eine Haskell-Funktion `insert :: Int -> Tree -> Tree`, die einen Integerwert in einen binären Suchbaum einfügt, so dass der resultierende Baum ebenfalls ein binärer Suchbaum ist.

(c) Definieren Sie unter Verwendung der Haskell-Funktion `insert` aus Teil (b) eine Haskell-Funktion `merge :: Tree -> Tree -> Tree`, die zwei binäre Suchbäume zu einem verschmilzt.