SoSe 2014

Prof. Dr. Margarita Esponda

ProInformatik II: Funktionale Programmierung

2. Übungsblatt fürs Wochenende

Abgabe: Montag um 8:55

Abgabe: Montag pünktlich um 8:55 Uhr vor Vorlesungsbeginn in **Papierform**.

1. Aufgabe (12 Punkte)

a) Definieren Sie mit Hilfe einer expliziten rekursiven Funktion einen (\\)-Infixoperator, der aus einer Liste **xs** alle Elemente entfernt, die sich in der Liste **ys** befinden.

Anwendungsbeispiel: [2, 6, 1, 7, 0, 9] \\ [2, 0, 9] => [6, 1, 7]

a) Definieren Sie die gleiche Funktion mit Hilfe von Listen-Generatoren

b) Definieren Sie mit Hilfe des (\\)-Operators, eine Funktion, die aus einer beliebigen Liste natürlicher Zahlen die kleinste natürliche Zahl findet, die nicht in der Liste vorkommt.

Anwendungsbeispiel: smallestNatNotIn [3, 5, 2, 7, 6, 10, 0, 1, 4, 12] => 8

c) Analysieren Sie die Komplexität Ihrer **smallestNatNotIn** Funktion.

2. Aufgabe (8 Punkte)

Betrachten Sie den folgenden algebraischen Datentyp, der rationale Zahlen darstellen soll.

```
data QRational = Q  ZInt  ZInt  deriving Show
```

Definieren Sie für den **QRational**-Datentyp folgende Funktionen:

qadd :: QRational -> QRational -> QRational

qmult :: QRational -> QRational -> QRational

qquotient :: QRational -> QRational -> QRational

qreciprocal :: QRational -> QRational

3. Aufgabe (6 Punkte)

Definieren Sie einen algebraischen Datentyp **Number**, mit dem Zahlen in Binär-, Oktal- und Hexadezimal-Darstellung modelliert werden.

Definieren Sie damit folgende Funktionen:

okt2Bin :: Number -> Number

Bin2Hex :: Number -> Number

okt2Hex :: Number -> Number

Achten Sie auf Fehlerbehandlung.

4. Aufgabe (8 Punkte)

a) Schreiben Sie eine Funktion, die in einer beliebig großen Zahl **z** mit **n** Ziffern das größte Produkt findet, das **k** nebeneinander stehende Ziffern haben. Wenn **k>n** ist, soll die Funktion einen Fehler zurückgeben.

b) Analysieren Sie die Komplexität der Funktion.

5. Aufgabe (8 Punkte)

Betrachten Sie den folgenden algebraischen Datentyp **ABaum**, der, anders als die Binärbaume, nicht nur zwei Kinder, sondern beliebig viele Kinder haben darf.

```
data ABaum a = Nil | Node a [ABaum a]
                                    deriving (Show, Eq)
```

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

a) **children** -- zählt die Anzahl der Kinder.
b) **depth** -- findet die Tiefe eines Baumes.
c) **find** -- Suchoperation
d) **mapTree** -- eine map-Funktion für den Baum

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.

2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

3) Kommentieren Sie Ihr Programme.

4) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.

5) Schreiben Sie in allen Funktionen die entsprechende Signatur.