WS 2018/2019 Funktionale Programmierung
5. Übungsblatt
M. Esponda/R. Rojas/O. Wiese
Abgabe: 3.12.2018 10:00
(Ohne Korrektur)

1. Aufgabe (3 Punkte)

Recherchieren Sie die Bindungssträke und die Assoziation von arithmetischen und logischen Operationen. Welche Bindungsstärke haben Funktionen?
Beschreiben Sie bitte Ihren Rechercheweg und geben Sie Ihre Quellen an.

2. Aufgabe (15 Punkte)

Im Lambda-Kalkül wurden die natürlichen Zahlen rekursiv als Nachfolger definiert. Übertragen Sie diese Konstruktion in Haskell. Gehen Sie dazu wie folgt vor:

1. Geben Sie einen neuen rekursiven Datentypen Nat an.
2. Definieren Sie zwei Funktionen zur Umwandlung von Nat in Int und umgekehrt. Die Funktionen sollten folgende Signaturen haben **toInt :: Nat -> Int** und **fromInt :: Int -> Nat**
3. Definieren Sie die aus dem Lambda-Kakül bekannten Funktionen:
    1. **isZero :: Nat -> Bool**
    2. **add :: Nat -> Nat -> Nat**
    3. **mult :: Nat -> Nat -> Na**
    4.

3. Aufgabe (7 Punkte)

Definieren Sie einen Operator für die Berechnung des größten gemeinsamen Teilers (ggT). Verwenden Sie die Symbole **.|.** zu Darstellung des Operators. Nutzen Sie zur Bestimmung vom ggT den klassischen euklidischen Algorithmus.
Sie finden eine Beschreibung des Algorithmus auf Wikipedia: https://de.wikipedia.org/wiki/Euklidischer_Algorithmus

4. Aufgabe (10 Punkte)

In dieser Aufgabe sollen Sie einige Funktionen auf Listen definieren.
Verwenden Sie dafür nur die Operatoren : und ++, sowie das Rekursionsprinzip. Sie dürfen Funktionen aus der Vorlesung verwenden, aber keine weiteren. Geben Sie jeweils die entsprechende Signatur an. Die Funktionen außer isPalindrom sollen für möglichst allgemeine Typen sein.
Implementieren Sie die folgenden fünf Funktionen auf Listen in Haskell (siehe Rückseite):

1. Die Funktion **abflachen**, welche eine Liste von Listen erhält und alle diese Listen in einer neuen Liste hintereinander hängt.
   Beispiel: abflachen[[1,2,3], [4,5], [6]] wird zu [1,2,3,4,5,6] ausgewertet.

2. Die Funktion **ohneLetztes**, welches eine Liste erhält und eine neue Liste mit allen Elementen aus der Eingabe ohne das letzte Element zurückgibt.
   Beispiel: ohneLetztes [1,2,3,4] wird zu [1,2,3] ausgewertet.

3. Die Funktion **isPalindrom**, die eine Liste von Zahlen erhält und prüft, ob die Zahlenfolge beim Rückwärtslesen und Vorwärtslesen gleich ist, also ein Palindrom.
   Beispiele: isPalindrom [3,5,3] wird zu True ausgewertet. isPalindrom [1,2] wird zu False ausgewertet.
   Hinweis: Diese Funktionen sollte nur für ganze Zahlen (Int) definiert werden. Verwenden Sie vorherige Funktionen und bereits in der Vorlesung definierte Funktionen.

4. Die Funktion **tupelize** erhält zwei Listen und macht daraus eine Liste von Tupeln aus den Elementen der beiden Listen.
   Beispiel: tupelize [1,2,3,4,5,6,7,8,9] [True, False] wird zu [(1,True), (2,False)] ausgewertet.

5. Die Funktion **detupelize** erhält eine Liste von Tupeln und soll ein Tupel von zwei Listen mit den jeweiligen Elementen zurückgeben. Sie ist quasi die “Umkehrfunktion” von tupelize.
   Beispiel: detupelize [(1, True), (2, False)] wird zu ([1,2], [True, False]) ausgewertet.

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 Ihre Programme.
4) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.
5) Schreiben Sie in alle Funktionen die entsprechende Signatur.