GOETHE UNIVERSITÄT
FRANKFURT AM MAIN
Fachbereich Informatik und Mathematik
Institut für Informatik

Vorsemesterkurs Informatik

Sommersemester 2011

Aufgabenblatt Nr. I.3

Aufgabe 1 (Listen)

a) Implementieren Sie in Haskell eine Funktion `zweites :: [a] -> a`, die das zweite Element einer Liste liefert. Hat die Eingabeliste weniger als zwei Elemente, so sollte eine Fehlermeldung generiert werden.

b) Implementieren Sie in Haskell eine Funktion `ohneLetztes :: [a] -> [a]`, die für eine Liste der Länge n die ersten $n-1$ Elemente der Liste (als Liste) liefert. Für den Fall $n = 0$ soll `ohneLetztes` eine Fehlermeldung ausgeben.

c) Implementieren Sie in Haskell eine Funktion `vertausche::[a] -> [a]`, die als Eingabe eine Liste erhält und je zwei nebeneinander liegende Elemente vertauscht. Z.B. soll `vertausche [1,2,3,4,5,6]` als Ergebnis die Liste `[2,1,4,3,6,5]` liefern.

Aufgabe 2 (Listen und Tupel)

Die Zuckerraffinerie Rohr & Rübe hat verschiedene Lieferungen Zuckerrüben und Zuckerrohr bekommen. Die Daten einer Lieferung bestehen aus dem Paar

$$(\text{Menge in Tonnen}, \text{Preis pro Tonne in EUR})$$

Definieren Sie in Haskell eine Funktion, die als Eingabe eine Liste von Lieferungen erhält und den Durchschnittspreis pro Tonne Zuckerrohstoff berechnet.

Hinweis: Definieren Sie zunächst eine Funktion, die das Paar (Gesamtmenge, Gesamtwert) berechnet.

Aufgabe 3 (Strings, Rekursion)

Ein *Palindrom* ist eine Zeichenkette die vorwärts und rückwärtsgelesen, dass gleiche Wort ergibt, z.B. ANNA oder RENTNER. Implementieren Sie in Haskell eine *rekursive* Funktion `istPalindrom :: String -> Bool`, die eine Zeichenkette als String erwartet und `True` oder `False` liefert, jenachdem, ob die Eingabe ein Palindrom ist oder nicht.

Aufgabe 4 (Strings)

Implementieren Sie in Haskell eine Funktion, die einen Text als String erhält und alle Zeilen mit Zeilennummern versieht und den nummerierten Text als Ausgabe liefert.

Hinweis: Mit der vordefinierten Funktion `show` können Sie eine Zahl in einen String konvertieren.

Aufgabe 5 (Listen und Typen)

Definieren Sie in Haskell je eine Funktion, die den folgenden Typ besitzt:

- `[Bool]`
- `Bool -> [[Integer]]`
- `[Integer] -> [Integer]`
- `[[[a]]] -> a`
- `Bool -> [Bool -> Bool]`

Aufgabe 6 (Imperatives Programmieren: Interpreter testen)

In der Datei `ImpInterpreter.hs` finden Sie einen Interpreter für imperative Programme, die in etwa den Sprachumfang aus Kapitel 4 des Skripts umfassen. Laden Sie die Datei in den GHCi und rufen Sie die Funktion `main` auf (dies startet den Interpreter für imperative Programme), machen Sie sich mit der Hilfe (Kommando `:help`), dem Sprachumfang (Kommando `:proghelp`) und dem Laden von imperativen Quellcodedateien (Kommand `:load`) vertraut.

Testen Sie den Interpreter zunächst mit den Befehlen:

```
X := 1;
X := X+1;
Z := Y;
new Y[10];
FOR (X:=0; X < 10; X:=X+1) { Y[X]:=X; };
X := 4;
WHILE X < 10 {Y[X] := Y[10-X] < 4; X:=X+1;}
```

Geben Sie die Befehle nacheinander ein, und lassen Sie sich nach jedem Befehl den *Zustand* anzeigen (mit dem Kommando `:state`).

Schreiben Sie anschließend das imperative Programm:

```
new Y[10];
FOR (X:=0; X < 10; X:=X+1) { Y[X]:=X; };
X := 4;
WHILE X < 10 {Y[X] := Y[10-X] < 4; X:=X+1;}
```

in eine Textdatei und laden Sie mit `:load` in den Interpreter für imperative Programme.

Aufgabe 7 (Imperatives Programmieren)

a) Gegeben seien drei Programmvariablen X_1, X_2 und X_3. Schreiben Sie ein imperatives Programm, dass die Werte der drei Variablen „rotiert“, d.h. X_1 hat danach den Wert von X_2, X_2 hat danach den Wert von X_3, und X_3 hat danach den Wert von X_1.

b) Sei X ein Array der Größe N. Implementieren Sie ein imperatives Programm, dass die Inhalte der N-Plätze im Array umdreht. Z.B. soll das Array ($N = 5$)

	0	1	2	3	4
X	10	2	33	54	12

nach Ablauf des Algorithmus die Form

	0	1	2	3	4
X	12	54	33	2	10

besitzen. Testen Sie Ihr Programm mit dem Interpreter für imperative Programme.