Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, D. Meier

Aufgabe 3 (Auswertungsstrategie): (24 Punkte)

Gegeben sei das folgende Haskell-Programm:

```
first::[Int] -> Int
first [] = 0
first (x:xs) = x

tillN::Int -> [Int]
tillN n = if (n <= 0) then []
          else n : tillN (n-1)

greaterThan::[Int] -> Int -> [Int]
greaterThan [] _ = []
greaterThan (x:xs) n = if(x > n) then x : (greaterThan xs n)
                       else (greaterThan xs n)
```

Die Funktion `first` gibt zu jeder Eingabeliste mit mindestens einem Element den ersten Eintrag der Liste zurück. Im Falle der leeren Liste wird `0` zurückgegeben. Die Funktion `tillN` erzeugt die absteigende Liste aller positiven ganzen Zahlen beginnend mit dem Eingabewert. Bei nicht-positiven Eingaben wird die leere Liste zurückgegeben. Beispielsweise erhält man beim Aufruf `tillN 3` die Liste `[3, 2, 1]` als Rückgabe. Die Funktion `greaterThan` erhält als Eingabe eine Integer–Liste `xs` und eine ganze Zahl `n`. Der Rückgabewert ist die Liste, die sich ergibt, wenn man alle Einträge löscht, die kleiner oder gleich `n` sind. Rufen wir `greaterThan [-1, 7, 0, 5, -9] 0` auf, so ergibt sich `[7, 5]`.
Geben Sie alle Zwischenschritte bei der Auswertung des Ausdrucks

```
first (greaterThan (tillN 1) (first []))
```

an. Unterstreichen Sie vor jedem Auswertungsschritt den Teil des Ausdrucks, der als Nächstes an seiner äußersten Position ausgewertet wird. Schreiben Sie hierbei (um Platz zu sparen) `f`, `tN` und `gT` statt `first`, `tillN` und `greaterThan`.

Hinweise:

- Beachten Sie die Hinweise zu Aufgabe 2.

Aufgabe 5 (Listen in Haskell): (5 + 5 + 4 + 5 + 5 = 24 Punkte)

Seien `x`, `y` und `z` ganze Zahlen vom Typ `Int` und seien `xs` und `ys` Listen der Längen n und m vom Typ `[Int]`. Welche der folgenden Gleichungen zwischen Listen sind richtig und welche nicht? Begründen Sie Ihre Antwort. Falls es sich um syntaktisch korrekte Ausdrücke handelt, geben Sie für jede linke und rechte Seite auch an, wie viele Elemente in der jeweiligen Liste enthalten sind und welchen Typ sie hat.

a) `((x:[]) : []) : [] = ((x : []) : []) ++ []`

b) `[x] ++ (y : ys) = x : ([y] ++ ys)`

c) `(x : (y : (z : []))) = (x : y) : (z : [])`

d) `(z : ys) : (xs : []) = [[z] ++ ys] ++ [xs]`

e) `[x,y] : [[z]] = (x : (y : [z])) : []`

Hinweise:

- Beachten Sie die Hinweise zu Aufgabe 4.

Aufgabe 8 (Programmieren in Haskell): (8 + 8 + 4 + 6 + 8 + 10 + 8 = 52 Punkte)

In dieser Aufgabe geht es um Matrizen über `Int`-Werten, die wir zeilenweise als Listen vom Typ `[Int]` repräsentieren. Eine Matrix wird also als Liste solcher Listen repräsentiert, sie hat den Typ `[[Int]]`. Als Beispiel betrachten wir die Einheitsmatrix mit Dimension 2, die in dieser Aufgabe von der Liste `[[1,0],[0,1]]` repräsentiert wird. Die Nummerierung der Zeilen und Spalten soll dabei stets mit 0 beginnen.
Implementieren Sie alle der im Folgenden beschriebenen Funktionen in Haskell. Geben Sie jeweils auch die Typdeklarationen an. Sie dürfen die Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), die Listenkonkatenation `++`, Vergleichsoperatoren wie `<=`, `==`, ..., boolesche Funktionen wie `&&`, `||`, `not` und die arithmetischen Operatoren `+`, `*`, `-` verwenden, aber **keine** vordefinierten Funktionen außer `length` und `!!`: Die Funktion `length` wird auf einer Liste ausgewertet und gibt deren Länge zurück. Die Funktion `!!` wird auf einer Liste und einem Index ausgewertet und gibt das Element der Liste an der durch den Index angegebenen Stelle zurück. Beispielsweise ergibt also `[10,11,12] !! 1` den Wert `11`. Schreiben Sie ggf. Hilfsfunktionen, um sich die Lösung der Aufgaben zu vereinfachen. Sie dürfen in allen Teilaufgaben Funktionen aus vorangegangenen Teilaufgaben nutzen.

a) `isMatrix matrix`
Gibt zurück, ob `matrix` vom Typ `[[Int]]` tatsächlich die Repräsentation einer Matrix ist. So soll `isMatrix [[1,0],[0,1]]` zu `True` auswerten, während `isMatrix [[1],[2,3]]` zu `False` auswerten soll.

b) `dimensions matrix`
Gibt die Dimensionen von `matrix` zurück, wobei `matrix` den Typ `[[Int]]` hat. Wenn `matrix` keine Matrix ist, soll `(-1,-1)` zurückgegeben werden. So soll `dimensions [[1,0],[0,1],[3,4]]` zu `(3,2)` auswerten.

Hinweise:

- Schreiben Sie je eine Hilfsfunktion für die Anzahl der Zeilen und die Anzahl der Spalten.

c) `isQuadratic matrix`
Gibt zurück, ob `matrix` vom Typ `[[Int]]` eine quadratische Matrix ist.

d) `getRow matrix i` und `getCol matrix i`
Gibt die `i`-te Zeile bzw. Spalte von `matrix` zurück, wenn `matrix` vom Typ `[[Int]]` eine Matrix ist. Ansonsten soll eine leere Zeile bzw. Spalte zurückgegeben werden. Auf Indizes `i`, die die Zeilen- bzw. Spaltendimension übersteigen oder negativ sind, darf sich die Funktion beliebig verhalten.

e) `trav matrix`
Gibt die transponierte Matrix zur Matrix `matrix` vom Typ `[[Int]]` zurück. Beim Transponieren einer Matrix wird diese entlang der ersten Hauptdiagonalen gespiegelt, d.h. jeder Eintrag a_{ij} der transponierten Matrix entspricht dem Eintrag a_{ji} der ursprünglichen Matrix. Wenn `matrix` keine Matrix ist, darf sich die Funktion beliebig verhalten.

f) `setEntry matrix i j aij`
Gibt diejenige Matrix zurück, die entsteht, wenn der Eintrag der Matrix `matrix` vom Typ `[[Int]]` in Zeile `i` und Spalte `j` auf den Wert `aij` gesetzt wird und alle anderen Werte von `matrix` gleich bleiben. Wenn `matrix` keine Matrix ist oder es die Position (i, j) in der Matrix nicht gibt, darf sich die Funktion beliebig verhalten.

g) Überlegen Sie, welche weitere Funktion auf Matrizen sinnvoll wäre und implementieren Sie diese. Wählen Sie eine Funktion aus, die nicht zu ähnlich zu einer der obigen Teilaufgaben ist. Beschreiben Sie in einem Kommentar kurz, was Ihre Funktion tun soll.