RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

Lehrstuhl für Hochleistungsrechnen
RWTH Aachen · D-52074 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Christian Bischof, Ph.D.
Andre Vehreschild, Jakob T. Valvoda

Übung *Programmierung WS 06/07* – Blatt 12

Lösungen müssen bis zum **29. Januar 2007, 17:00 Uhr** in den Kasten Ihrer Übungsgruppe eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe**, sowie die **Namen** und **Matrikelnummern** der Mitglieder Ihrer 2er-Gruppe auf jedes Lösungsblatt Ihrer Abgabe zu schreiben. Bitte heften Sie ihre Blätter zusammen.

Alle erstellten Haskell-Programme sind sowohl in gedruckter Form abzugeben, als auch per E-Mail an den jeweiligen Tutor zu senden. Vergessen Sie auch bei der elektronischen Abgabe per E-Mail nicht die Angabe der **Nummer Ihrer Übungsgruppe**, sowie die jeweiligen **Namen** und **Matrikelnummern**.

Aufgabe 1 (1 + 0.5 + 1.5 + 1 = 4 Punkte)

Geben Sie den allgemeinsten Typ der folgenden Haskell-Funktionen und Ausdrücke an. Begründen Sie Ihre Antwort in eigenen Worten! Für die reine Angabe des Typs ohne eine Begründung bekommen Sie keine Punkte!

a) `f x y z = z : (f y x z)`

b) `snoc x y = y ++ [x]`

c) `curry (\(x,y) z -> (\u v w -> (f u v w)) x y)`

d) `(\x y z -> x y z)`

Hinweis: Beachten Sie, dass in c) die Funktion `f` des Aufgabenteils a) verwendet wird.

Aufgabe 2 (3 + 3 = 6 Punkte)

In dieser Aufgabe sollen Sie analog zu Übung 6, Aufgabe 4 einen Stack und eine Konversionsroutine in Haskell implementieren. Ein Stack sei in Haskel als `data Stack a = Nil | Top a (Stack a)` deklariert. Folgende Ausdrücke ergeben dementsprechend einen Stack

- `Nil` ergibt den leeren Stack.
- `Top 5 Nil` ergibt einen `Int`-Stack mit dem Element `5`.
- `Top '+' (Top '*' (Top '-' Nil))` ist vom Typ `Stack Char` und enthält die Elemente `'+'`, `'*'` und `'-'`, wobei `'+'` das oberste Element darstellt.

a) Implementieren Sie folgende Funktionen zur Modifikation des Stacks:

- `pop :: Stack a -> Stack a`, entfernt das oberste Element vom Stack und *liefert den modifizierten Stack zurück.* der Aufruf von `pop Nil` resultiert ebenfalls im leeren Stack `Nil`.
  Beispiel: `pop (Top '+' (Top '*' (Top '-' Nil)))` → `Top '*' (Top '-' Nil)`
- `push :: a -> Stack a -> Stack a`, bekommt ein neues Element und den Stack als Argumente und lifert den modifizierten Stack.
  Beispiel: `push '-' Nil` → `Top '-' Nil`
- `top :: Stack a -> a`, liefert das oberste Element vom Stack. Das Ergebnis ist für einen leeren Stack (`Nil`) nicht definiert.
  Beispiel: `top (Top '+' (Top '*' (Top '-' Nil)))` → `'+'`

b) Implementieren Sie eine Funktion `convert :: String -> String`, welche einen Ausdruck in Infix-Notation in einen Ausdruck in Postfix-Notation umwandelt und dabei die obigen Stack-Funktionen nutzt. Der Ausdruck sollte zuerst von allen unzulässigen Zeichen bereinigt werden. Insgesamt sollen nur einstellige Zahlen (Ziffern), die Operatoren `+`, `-`, `*` und `/` und runde Klammern erlaubt sein. Der Infix-Ausdruck muss vollständig geklammert sein. Das Ergebnis ist nur für korrekt und vollständig geklammerte Infix-Ausdrücke definiert. Vgl. Sie hierzu die Beispiele aus Übung 6, Aufgabe 4.

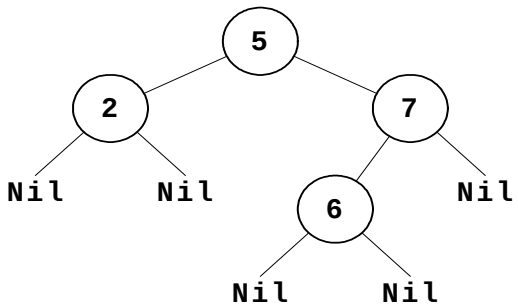
Hinweis: Sie können beliebige Hilfsfunktionen definieren. Verwenden Sie ebenfalls Listen und die Funktion `elem`, um festzustellen, welche Bedeutung ein Zeichen hat. Seien z. B. `operator = "+-*/"` und `ziffer = "0123456789"` zwei solche Listen und `x` ein Zeichen. `elem x operator` wird wahr, wenn `x` ein Operator ist. Entsprechend ist `elem x ziffer` wahr, wenn `x` eine Ziffer ist.

Aufgabe 3 (3 + 5 = 8 Punkte)

Gegeben sei die folgende Deklaration eines binären Baums in Haskell

```
data BinTree a = Nil | Node a (BinTree a) (BinTree a) deriving Show
```

Dieser ist entweder leer (`Nil`) oder besteht aus einem Knoten, welcher einen linken und einen rechten Teilbaum enthält. Zum Beispiel repräsentiert der Ausdruck `Node 5 (Node 2 Nil Nil) (Node 7 (Node 6 Nil Nil) Nil)` einen binären Baum und kann graphisch wie folgt dargestellt werden:



a) Untersuchen Sie die folgenden Haskell-Ausdrücke. Falls es sich um einen binären Baum handelt, dann zeichnen Sie den Baum in der obigen graphischen Repräsentation. Vergessen Sie hierbei nicht die Angabe der Blätter (`Nil`). Sollte der Ausdruck keinen binären Baum beschreiben, dann geben Sie die Fehlerstelle an und beschreiben Sie kurz den Fehler.

- `Node "Wurzel" (Node Nil "Blatt" Nil)`
- `Node 3 (Node 1 Nil Nil) (Node 8 (Node 7 (Node 6 (Node 5 Nil Nil) Nil) (Node 7 Nil Nil)) (Node 9 Nil Nil))`
- `Node 2 (BinTree 3) (BinTree 4)`
- `Node "Node" Nil (Node "BinTree" (Node "Left" Nil (Node "Nil" Nil (Node "Right" Nil Nil))) Nil)`

Binäre Suchbäume sind binäre Bäume, deren Elemente sortiert sind (vergleichen Sie hierzu Übung 8, Aufgabe 2). Gegeben sei die folgende Haskell-Funktion, welche eine ganze Zahl sortiert in einen binären Suchbaum einfügt:

```
insert :: Int -> BinTree Int -> BinTree Int
-- einfuegen in leeren Baum erzeugt neuen Knoten
insert x Nil = Node x Nil Nil
-- fuege neues Element sortiert in jeweiligen Teilbaum ein
insert x (Node y left right)
  | x < y     = Node y (insert x left) right -- fuege links ein
  | otherwise = Node y left (insert x right) -- fuege rechts ein
```

b) Implementieren Sie analog zur Funktion `insert` folgende Funktionen:

- `insertList :: [Int] -> BinTree Int -> BinTree Int`, welche eine Liste von ganzen Zahlen und einen binären Suchbaum übergeben bekommt, und alle Elemente der Liste in diesen Baum einfügt. Die Funktion resultiert in dem modifizierten binären Suchbaum.
- `insertTree :: BinTree Int -> BinTree Int -> BinTree Int`, welche zwei binäre Suchbäume übergeben bekommt, und alle Elemente des ersten Suchbaums in den zweiten einfügt. Der modifizierte zweite Suchbaum wird zurückgegeben.
- `search :: Int -> BinTree Int -> Bool`, welche eine ganze Zahl und einen binären Suchbaum übergeben bekommt und überprüft, ob diese Zahl im Suchbaum enthalten ist. Die Funktion liefert `True`, falls das Element gefunden wurde und `False` sonst.
- `delete :: Int -> BinTree Int -> BinTree Int`, welche eine ganze Zahl und einen binären Suchbaum übergeben bekommt und nach dem ersten Vorkommen der Zahl sucht. Wird ein solcher Knoten gefunden, dann wird dieser durch die beiden Teilbäume des Knotens ersetzt. Diese müssen wiederum geeignet zu einem binären Suchbaum zusammengefügt werden.
- `toList :: BinTree Int -> [Int]`, welche einen binären Suchbaum übergeben bekommt und aus den Elementen des Suchbaums eine sortierte Liste erzeugt. Nutzen Sie hierbei die Eigenschaft des binären Suchbaums aus, dass die einzelnen Elemente bereits sortiert im Baum gespeichert sind.

Aufgabe 4 (2 + 2 = 5 Punkte)

Gegeben seien die Java-Programme P_1 und P_2. Vervollständigen Sie die Verifikation der Algorithmen im Hoare-Kalkül, indem Sie die unterstrichenen Teile (falls erforderlich) ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur dann eine Zusicherung stehen, wenn sie aus einer Regel des Hoare-Kalküls folgt. Geben Sie bei jedem Schritt den Namen der verwendeten Hoare-Regel an.

a) Der Algorithmus P_1 vertauscht den Inhalt der Variablen x und y.

```
x = x + y;
y = x - y;
x = x - y;
```

Geben Sie bei der Verifikation von P_1 zuerst eine geeignete Nachbedingung an!

$\langle\ x = \mathtt{A} \wedge y = \mathtt{B}\ \rangle$

$\langle$ _____ $\rangle$

`x = x + y;`

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

`y = x - y;`

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

`x = x - y;`

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

b) Der Algorithmus P_2 berechnet ausgehend von einer Zahl a die nächst größere gerade Zahl.

```
if( (a % 2) != 0 )
    r = a+1;
else
    r = a
```

Gehen Sie bei der Vervollständigung der Verifikation von den nachfolgenden Vor- und Nachbedingungen aus.

$\langle$ true $\rangle$

$\langle$ _____ $\rangle$

```
if( (a % 2) != 0 )
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
   r = a+1;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
else
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
   r = a;
```

$\langle$ _____ $\rangle$

$\langle$ r $= 2 \cdot$ n $\rangle$