# 4. Übungsaufgabe zu
# Fortgeschrittene Funktionale Programmierung
# Thema: Monadische Programmierung
# ausgegeben: 27.05.2005, fällig: 03.06.2005

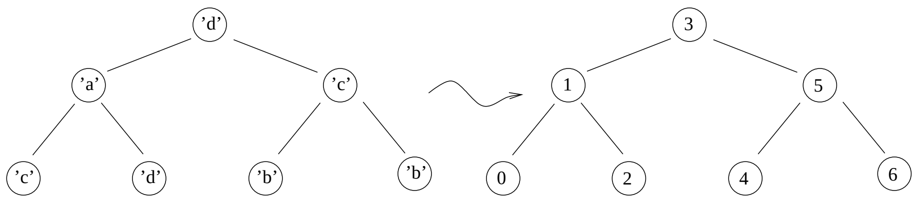
Für dieses Aufgabenblatt sollen Sie einige Papier- und Bleistiftaufgaben lösen und eine Reihe von Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln. Legen Sie Ihre Lösungen für die Abgabe in einer Datei namens `AufgabeFFP4.hs` in Ihrem Homeverzeichnis wie gewohnt auf oberstem Niveau ab. Kommentieren Sie Ihre Programme wie immer aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. Die Lösung der Papier- und Bleistiftaufgaben können Sie in der Datei `AufgabeFFP4.hs` in der Form eines Haskell-Kommentars angeben.

*Programmieraufgaben*:

- Wir betrachten den Datentyp

```
data Tree a = Leaf a | Node a (Tree a) (Tree a)
    deriving Show
```

  - Schreiben Sie eine monadenfreie Haskell-Rechenvorschrift `label :: Tree a -> Tree Integer`, die einen Argumentbaum in Infix-Ordnung durchläuft und dabei die Knoten und Blätter beginnend mit 1 fortlaufend durchnummeriert. Das nachstehende Beispiel illustriert die gewünschte Funktionalität:




  - Schreiben Sie entsprechend eine Haskell-Rechenvorschrift `mlabel :: Tree a -> Tree Integer`, die die Aufgabenstellung aus der vorigen Teilaufgabe mithilfe monadischer Programmierung löst.

*Papier- und Bleistiftaufgaben*:

- Betrachten Sie die Aufgabe von Folie 61 zu Vorlesungsteil 5. Zeigen Sie mit Hilfe der Definitionen von `>>=`, `map`, `concat`, `return`, etc., dass der Ausdruck

```
[1,2,3] >>= \x -> [4,5,6] >>= \y -> return (x,y)
```

und daher auch der Ausdruck

```
do x <- [1,2,3]; y <- [4,5,6]; return (x,y)
```

den Wert

```
[(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)]
```

hat.

- Betrachten Sie Folie 64 und die Folien 68 und 69 zu Vorlesungsteil 5 und damit folgende Situation:

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)
  deriving Show

newtype Label a = Label (Integer -> (Integer,a))

instance Monad Label where
  return a
    = Label (\s -> (s,a))
  Label lt0 >>= flt1
    = Label $ \s0 ->
        let (s1,a1)   = lt0 s0
            Label lt1 = flt1 a1
        in lt1 s1

mlabel   :: Tree a -> Tree Integer
mlabel t = let Label lt = mlab t
           in snd (lt 0)

mlab :: Tree a -> Label (Tree Integer)
mlab (Leaf a)
     = do n <- getLabel
          return (Leaf n)
mlab (Branch t1 t2)
     = do t1' <- mlab t1
          t2' <- mlab t2
          return (Branch t1' t2')

getLabel :: Label Integer
getLabel = Label (\n -> (n+1,n))
```

Rechnen Sie nach, dass der Ausdruck `mlabel (Leaf 'c')` den Wert `Leaf 0` hat.

Fügen Sie die Lösung der beiden Papier- und Bleistiftaufgaben der Datei `AufgabeFFP4.hs` in Form eines Haskell-Kommentars bei.