```haskell
-- Tut exercises on trees
--
-- Copyright [2001..2002] Keller & Chakravarty

module TutTrees where

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

-- if we find the node and the left subtree t1 is empty,
-- we simply remove the topmost node
-- otherwise. we move the topmost node t1 up one level
-- and delete it from t1. In general, this will destroy the
-- ordering of the tree
deleteFromTree:: Ord a => a -> Tree a -> Tree a
deleteFromTree e Leaf = Leaf
deleteFromTree e (Node x  t1 t2)
  | e < x = Node x (deleteFromTree e t1) t2
  | e > x = Node x t1  (deleteFromTree e t2)
  | otherwise =
      case t1 of
        Leaf            -> t2
        Node y t1' t2'  -> Node  y (deleteFromTree y t1) t2


-- 2nd Exercise ------------------------------------------------------------

data  Tree1 a = Leaf1 a
              | Node1 a (Tree1 a) (Tree1 a)
              deriving (Show)

-- Tree1 and Tree are not equivalent, as we cannot map
-- Leaf :: Tree to any value of type Tree1, i.e., Tree1
-- cannot model an emptry tree.
--
-- Tree [a] and [a]:
-- Although we can define functions from [a] -> Tree a
-- (for example, similar to insertListTree) and
-- Tree a -> [a] (e.g) flatten, operations of latter type
-- always loose information, namely the structure of the tree
-- so that different trees have to be mapped to the same list
-- so we cannot reconstruct the original tree from the list.
--
-- (the students should understand that names are not important,
--  but the number of constructors, their arity and the type of
--  their parameters)
```