Folding Trees

Recall how we defined trees in the past:

```
data Tree a = E | N (Tree a) a (Tree a)
```

It is natural for us to want to traverse the trees. The most universal way to do so is to define folding functions analogous to foldr or foldl. We will need three such functions, as trees can be traversed in three ways:

Inorder With *inorder traversal*, the nodes on the left child are visited first, then the root, then the nodes on the right child (left-root-right).

Preorder With *preorder traversal*, the root is visited first, then the nodes on the left child, then the ones on the right child (root-left-right).

Postorder with *postorder traversal*, the nodes on the left child are visited first, then the ones on the right child, and finally the root (left-right-root).

Let's take a look at how we can implement each of these:

```
foldin :: (a -> b -> b) -> Tree a -> b -> b
foldin _ E v                = v
foldin f (N left x right) v = v3
    where v1 = foldin f left v
          v2 = f x v1
          v3 = foldin f right v2
```

We could actually also write these in a "point-free" way, avoiding direct references to v:

```
foldin _ E              = id          -- The identity function
foldin f (N left x right) = foldin f right . f x . foldin f left
```

Practice: Implement the other two traversals, foldpre and foldpost.