3. This question is about how to model *polymorphic finite binary trees* in Haskell using first order function definitions. Using the `data` declaration,

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

we can for example define,

```
    test :: Tree Char
    test = Node (Leaf 'a') 'b' (Node (Leaf 'd') 'c' (Leaf 'e'))
```

Define each of the following functions.

(a) A *show* function `showt :: Tree a -> [Char]` which represents an enumeration of the nodes in string format using a *left to right depth first* traversal of the nodes of a given tree. For example, `showt test` evaluates to `"abdce"`. [6]

(b) `depth :: Tree a -> Int` which returns the *depth* of a given tree. This being defined to be the length of a longest branch, which is defined to be the number of nodes in such a branch. For example, `depth test` evaluates to `3`. [6]

(c) `equalt :: (Tree a, Tree a) -> Bool` which returns `True` if & only if two given trees are *equal*. That is, if their nodes are equal, and if recursively their subtrees are equal. [8]