Recursive Data Structure: Tree

A binary tree consists of leaf nodes and branch (internal) nodes:

- A leaf node is just that, a leaf node.
- A branch node has two children, which in turn are trees recursively.

This is coded in Haskell as:

```
data Tree = Leaf | Branch Tree Tree
  deriving Show
```

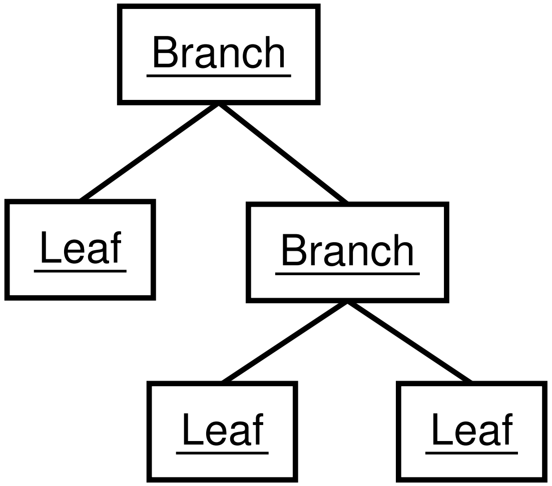
1. A leaf node is a tree.
2. A branch node with two tree-type children is also a tree.

Tree Example

Here is an example `Tree` expression:

```
Branch Leaf (Branch Leaf Leaf)
```

Here is how it looks like conceptually:



Polymorphic Trees

You may want to store some data in your trees. E.g., store numbers in leaf nodes:

```
data IntTree = IntLeaf Int | IntBranch IntTree IntTree
```

But if you do this, you may have to repeat it for other types of data:

```
data STree = SLeaf Shape | SBranch STree STree
data BoolTree = ...
```

Worse, if you need a function to compute, say, the number of nodes in a tree, you will have to write a separate version for each of the above tree types (due to strong typing).

How should you avoid such repetitions?

Polymorphic Trees

You should use polymorphism to avoid such repetitions.

```
data LTree a = ...
```

The type name is parameterized by the type variable a. The user will instantiate it to the actual data type stored in the tree.

```
data LTree a = LLeaf a | LBranch (LTree a) (LTree a)
```

A leaf takes a parameter of type a that is the datum to be stored.

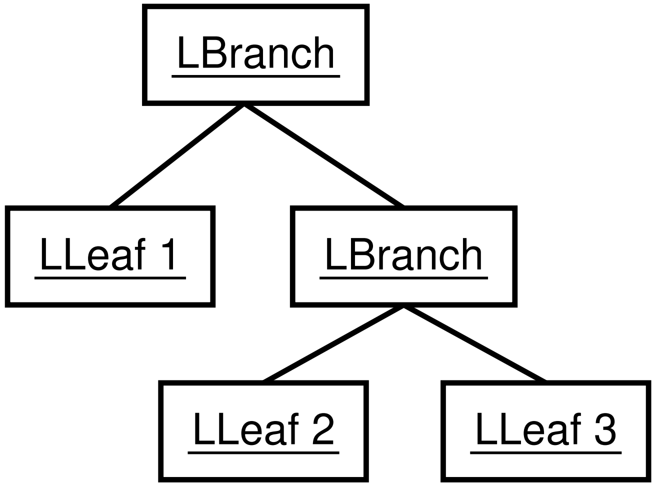
A branch takes two children as parameters. Note that the full type name `LTree a` must be used.

Polymorphic Tree Example

Example:

```
LBranch (LLeaf 1) (LBranch (LLeaf 2) (LLeaf 3))
  :: LTree Int
```

It looks like:



Functions for Polymorphic Trees

Write a function that counts the number of nodes (both leaves and branches) in a tree.

```
totalNumofNodes :: LTree a -> Int
```

The parameter type has the type variable a because we do not care what data are in the leaves.

```
totalNumofNodes (LLeaf _) = 1
totalNumofNodes (LBranch x y) =
  1 + totalNumofNodes x + totalNumofNodes y
```

More Polymorphic Trees

To stuff data into branch nodes instead (and no data at leaves):

```
data ITree a = ILeaf | IBranch a (ITree a) (ITree a)
```

To stuff data into both kinds of nodes:

```
data DTree a = DLeaf a | DBranch a (DTree a) (DTree a)
```

To stuff one type of data into branches and another type into leaves:

```
data FTree a b = FTree a
               | FBranch b (FTree a b) (FTree a b)
```