CS 456/556: Advanced Declarative Programming
Flipped Classroom Exercise 2

"A rose by any other name would smell as sweet." – William Shakespeare

Consider the following definition for *List*:

```
data List a = Nil | Cons {car :: a, cdr :: List a}
```

Given the above, solve the following problems:

1. Give *Functor* and *Monoid* instances for *List*.
2. Give an *Applicative* instance for *List* that doesn't use *ap* to define $<*>$.

   Note: Your definition of $<*>$ should combine two lists such that every function in the first list is applied to every value in the second exactly once. For example,

   ```
   > let f = Cons (+ 1) (Cons (+ 2) (Cons (+ 3) Nil))
   > let x = Cons 1 (Cons 2 (Cons 3 Nil))
   > f <*> x
   Cons 2 (Cons 3 (Cons 4 (Cons 3 (Cons 4 (Cons 5
     (Cons 4 (Cons 5 (Cons 6 Nil))))))))
   ```

3. Verify that *pure id* is a right and left unit of $<*>$.
4. Give a *Foldable* instance for *List*. Test your implementation as follows:

   ```
   > sum x
   6
   > and (Cons True (Cons False Nil))
   False
   > any even x
   True
   ```

5. Give *Monad* and *MonadPlus* instances for *List*.

Consider the following definition for *RoseTree*:

```
data RoseTree a = Node a [RoseTree a]
```

Given the above, solve the following problems:

1. Give *Functor* and *Applicative* instances for *RoseTree*. Note: Your definition of $<*>$ should combine two trees such that every function in the first tree is applied to every value in the second exactly once. For example,

```
> let f = Node (+ 1) [Node (+ 2) [], Node (+ 3) []]
> let x = Node 1 [Node 2 [], Node 3 []]
> f <*> x
Node 2 [Node 3 [],Node 4 [],Node 3 [Node 4 [],Node 5 []],
  Node 4 [Node 5 [],Node 6 []]]

> let g = Node (+ 1) [Node (+ 2) []]
> let y = Node 1 [Node 2 []]
> g <*> y
Node 2 [Node 3 [],Node 3 [Node 4 []]]
```

2. Verify that *pure id* is a right and left unit of $<*>$.

3. Now consider the following definition for a *Zippable* type class:

```
class Zippable z where
  zipWith :: (a -> b -> c) -> z a -> z b -> z c
  zip :: z a -> z b -> z (a,b)
  zip = zipWith (,)
  (<+>) :: z (b -> c) -> z b -> z c
  (<+>) = zipWith ($)

infixl 1 <+>
```

   Give a *Zippable* instance for *RoseTree.*

4. A new type *ZipRoseTree* can be defined as follows:

```
newtype ZipRoseTree a = ZipRoseTree {getZipRoseTree :: RoseTree a}
```

   Give an *Applicative* instance for *ZipRoseTree* where $<*>$ applies a tree of functions to a tree of values by zipping them. For example,

```
> ZipRoseTree g <*> ZipRoseTree y
ZipRoseTree {getZipRoseTree = Node 2 [Node 4 []]}
```

5. Verify that *pure id* is a right and left unit of $<*>$.