CS 456/556: Advanced Declarative Programming
Flipped Classroom Exercise 4

1. Consider the following definition of a parser combinator type:

   ```
   newtype Parser a = Parser {parse :: String -> Maybe (String, a)}
   ```

   - Make *Parser* an instance of type class *Monad*.
   - Make *Parser* an instance of type class *MonadPlus*.
   - Define a *Parser Char* called *item* that consumes a character.
   - Define a function called *sat* that takes a predicate *Char −> Bool* argument and returns a *Parser Char* that consumes a character if it satisfies the predicate and fails otherwise.
   - Define a function called *string* which takes a *String* argument and returns a *Parser String* that consumes a matching string and fails otherwise.

2. Consider the following definition of a binary tree type:

   ```
   data Btree a = Leaf a | Fork (Btree a) (Btree a) deriving Show
   ```

   The derived instance for *Show* will cause a *Btree* to be printed as Haskell code which could be used to construct it. For example,

   ```
   > Fork (Leaf 1) (Fork (Fork (Leaf 2) (Leaf 3)) (Leaf 4))
   Fork (Leaf 1) (Fork (Fork (Leaf 2) (Leaf 3)) (Leaf 4))
   ```

   Using the parser monad, define a *Parser (Btree Int)* which will construct a *Btree Int* from its printed representation. For example,

   ```
   > btree "Fork (Leaf 1) (Fork (Fork (Leaf 2) (Leaf 3)) (Leaf 4))"
   Fork (Leaf 1) (Fork (Fork (Leaf 2) (Leaf 3)) (Leaf 4))
   ```