

Extra parsing exercises

Parsing a simple grammar with ReadP/Parsec

Consider the following grammar of simple, additive arithmetic expressions:

```
E ::= E "+" T | E "-" T | T | "-" T .  
T ::= num | "(" E ")" .
```

Here, E is the start symbol, the token num consists of one or more decimal digits, and all tokens may be separated by arbitrary whitespace. For example, "-1 + 23 - (-456)" is a well formed expression. We want to parse such expressions into the following AST:

```
data Exp = Num Int | Negate Exp | Add Exp Exp
```

Note that a subtraction, as in $x - y$, is treated as syntactic sugar for an addition and negation, i.e., $x + (-y)$. Thus, the sample string above should be parsed as the following value of type Exp:



```
Add (Add (Negate (Num 1))  
         (Num 23))  
      (Negate (Negate (Num 456)))
```

We want to implement a Exp parser module with the following functionality:

```
type ParseError = ... -- should be an instance of at least Eq and Show  
  
parseString :: String -> Either ParseError Exp
```

That is, parseString s should return either an AST for the expression, or some human-readable error message (which may be as uninformative as "no parse").

Tasks:

1. The grammar above is left-recursive, and thus not directly suited for parsing with most combinator libraries. Rewrite it to an equivalent grammar without left recursion
2. Implement a parser for the grammar from (1) using the ReadP library. A stub implementation is available [here](#) .
3. (Optional, but strongly recommended if you want to use Parsec for assignments/exam) Implement a parser for the grammar from (1) using the Parsec library. A stub implementation is available [here](#) .