CSCC24 2020 Winter – Assignment 3
Due: Wednesday, March 25, midnight
This assignment is worth 10% of the course grade.

In this assignment, you will implement in Haskell a parser for a toy language.

As usual, you should also aim for reasonably efficient algorithms and reasonably organized, comprehensible code.

Expression Syntax

We will have a "simple" (just you wait) expression syntax. Here is the EBNF grammar, together with the informal points afterwards for completion and disambiguation.

```
<expr> ::= <literal>
        | <var>
        | <op1> <expr>
        | <expr> <op2> <expr>
        | "(" <expr> ")"

<op1> ::= "!" | "-"
<op2> ::= "+" | "-" | "==" | "&&" | "||"
```

Informal points:

- The start symbol is `<expr>`.
- `<literal>` is for natural number literals: One or more digits. (Unary prefix minus is handled separately.)
- `<var>` is for variable names: A letter followed by zero or more letters or digits. However, the following are reserved words and cannot be variable names: `or`, `assert`, `while`, `do`. (These are for constructs that will appear in the next assignment!)
- Ambiguity under `<expr>` is resolved by operator precedence and association. From lowest precendence to highest:

operator	association
`||`	right
`&&`	right
`==`	none, e.g., "`x == y == z`" is unexpected
`+`, infix binary `-`	left
`!`, prefix unary `-`	
literal, var, parentheses	

- Whitespaces around tokens are possible.

The abstract syntax tree is defined by these types:

```
data Expr = LitNat Integer
          | Var String          -- typo fixed Mar 19
          | Prim1 Op1 Expr
          | Prim2 Op2 Expr Expr
    deriving (Eq, Show)

data Op1 = Not | Neg
    deriving (Eq, Show)

data Op2 = Add | Sub | EqNat | And | Or
    deriving (Eq, Show)
```

Basically `Prim1` is for the unary operators, and `Prim2` is for the binary operators.

Implement a parser for Orlang. A main parser `mainParser` is already provided, so you can focus on the start symbol parser `expr :: Parser Expr` and downwards.

From the lectures, you already know how to implement infix operator precedence. The extra challenges in this assignment are `==` and the prefix unary operators.

In the case of the prefix unary operators, take care that these inputs are legal and their corresponding abstract syntax trees are:

input	AST
`- - 5`	`Prim1 Neg (Prim1 Neg (LitNat 5))`
`- - - 5`	`Prim1 Neg (Prim1 Neg (Prim1 Neg (LitNat 5)))`
`! - ! 5`	`Prim1 Not (Prim1 Neg (Prim1 Not (LitNat 5)))`

(Typo fixed March 20)

How do you get this to work?

End of questions.