CSCC24 Winter 2018 – Assignment 4
Due: March 24 Saturday, 18:00 (6PM)
https://markus.utsc.utoronto.ca/cscc24w18/
This Assignment is worth 10% of the course grade.

All of the following count: Correctness, simplicity, DRY (don't repeat yourself), efficiency, style.

[10 marks] Implement a parser for the following language. The language is defined by starting with the following grammar in EBNF, and then there are amendments and remarks afterwards.

```
block    ::= cond | lambda | let | infix
infix    ::= test { boolop test }
boolop   ::= "&&" | "||"
test     ::= arith [ cmp arith ]
cmp      ::= "==" | "/=" | "<" | "<="
arith    ::= { addend addop } addend
addop    ::= "+" | "-"
addend   ::= { factor mulop } factor
mulop    ::= "*" | "/"
factor   ::= { atom } atom
atom     ::= "(" block ")" | literal | var
cond     ::= "if" block "then" block "else" block
lambda   ::= "\" var "->" block
let      ::= "let" "{" { equation } "}" "in" infix
equation ::= var "=" block ";"
literal  ::= integer | boolean
boolean  ::= "True" | "False"
```

Amendments and remarks:

- The start symbol is block.
- Two non-terminals, integer and var, are specified informally by:
  - integer: An optional minus sign, followed by one or more decimal digits.
  - var: A letter, followed by zero or more letters or digits. However, reserved words are not allowed to be vars. The reserved words are:
    if, then, else, let, in, True, False.
    If the parser sees a reserved word when a var is expected, it is considered a parser error.
- There can be spaces, tabs, and newlines (collectively known as whitespaces) around integers, vars, and terminal strings. For example

```
   let { x
         =  4 * (
  5+6) ;
} in \ y ->  if  x< y then x else y
```

  is allowed.

  The best way to skip whitespaces is to adopt this strategy consistently. The entry-point parser consumes leading whitespaces once and for all. Note that aftewards you can think of all remaining whitespaces as trailing whitespaces. So parsers for integers, vars, and terminal strings consume trailing whitespaces. There are building blocks in ParserLib.hs that do this.

The parser's answer, if there is no error, is an abstract syntax tree represented by the Term type in A4Term.hs. Most of the expected answers should be self-evident. Here are a few non-obvious points:

- The boolean infix operators `&&`, `||` associate to the right.
- The arithmetic infix operators `+`, `-`, `*`, `/` associate to the left.
- Infix expressions parse to `Prim2` trees. Put the operator in the string field. Example: `5+6` is parsed to `Prim2 "+" (Num 5) (Num 6)`.
- The grammar rule
  `factor ::= { atom } atom`
  covers function application. When there are 2 or more atoms, the answer is an `App` tree. Function application associates to the left. For example `f x y` is parsed to `App (App (Var "f") (Var "x")) (Var "y")`.

  This rule is best implemented like `some atom`, where `atom` is your atom parser. Then `foldl1 App` finishes the job.

(End of questions.)