Expr.hs

module Main where

-- The import statements in this file include import lists, which state
-- exactly what is imported. This can be nice documentation, so that
-- readers know what comes from where.
-- These imports you know how to deal with.
import Data.Char ( isSpace, isDigit )
import Text.Read ( readMaybe )

-- These are more advanced, used only in 'main'.
import Control.Exception ( SomeException(..), evaluate, catch )
import Control.Monad     ( when )
import System.Exit       ( exitSuccess )

-- The AST type for parsed expression trees
data Expr
  = Plus Expr Expr
  | Minus Expr Expr
  | Times Expr Expr
  | Divide Expr Expr
  | Num Integer
  deriving (Eq, Show)

-- Possible tokens
data Token
  = PlusT
  | MinusT
  | TimesT
  | DivideT
  | NumT Integer
  deriving (Eq, Show)

-- Read an input string into a list of tokens.
lexTokens :: String -> [Token]
lexTokens input = lexNoPrefix (findToken input)

-- Drop any non-lexed prefix of the input. This language
-- is so simple that we can just use dropWhile.
findToken :: String -> String
findToken = dropWhile isSpace

-- Lex an input string, assuming that the first thing
-- in the string (if anything) is a token (as opposed to
-- whitespace).
lexNoPrefix :: String -> [Token]
lexNoPrefix []     = []
lexNoPrefix (c:cs) = token : lexTokens rest
  where
    (token, rest) = lex1 c cs

-- lex a number
lex1 :: Char -> String -> (Token, String)
lex1 'n' cs = (NumT n, rest)
  where
    (n, rest) = readMaybe (c:more_digs)

-- lex the operators
lex1 '"' cs = (PlusT, cs)
lex1 '\' cs = (MinusT, cs)
lex1 '"' cs = (TimesT, cs)
Expr.hs

73: lex1 '/' cs = (DivideT, cs)
74:
75: -- Otherwise, we have a lexical error
76: lex1 c cs = error ("No lex: " ++ (c:cs))
77:
78: -- Parse one expression from a list of tokens, also returning
79: -- the remaining, unparsed tokens.
80: parse1 :: [Token] -> (Expr, [Token])
81: parse1 = error "not implemented yet"
82:
83: -- Parse a list of tokens into an expression. Errors if there are
84: -- too few or too many tokens.
85: parse :: [Token] -> Expr
86: parse = error "not implemented yet"
87:
88: -- Evaluate an expression to a number.
89: eval :: Expr -> Integer
90: eval = error "not implemented yet"
91:
92: -- Evaluate a string into a number.
93: evalString :: String -> Integer
94: evalString str = eval (parse (lexTokens str))
95:
96: -- A read-eval-print loop (REPL)
97: -- (You are not expected to understand this.)
98: main :: IO ()
99: main = do
100:
101: -- primary user interaction commands
102: putStrLn ""
103: putStrLn "Enter a prefix expression:"
104: expr_string <- getLine
105:
106: -- allow users to quit
107: when (expr_string == "quit")
108: exitSuccess
109:
110: -- This code runs evalString in a way that, if evalString calls 'error',
111: -- the program will not immediately abort. The Haskell features used here
112: -- are beyond the scope of CS245. The curious may enjoy looking these
113: -- functions up online.
114: catch (do value <- evaluate (evalString expr_string)
115:         print value)
116:     (\ (SomeException e) -> print e)
117:
118: -- And do it again.
119: main