Expr.hs

1: {- Author: Richard Eisenberg, edited by <your name here>
2:     File: Expr.hs
3: }
4: module Main where
5:
6: -- The import statements in this file include import lists, which state
7: -- exactly what is imported. This can be nice documentation, so that
8: -- readers know what comes from where.
9: -- These imports you know how to deal with.
10: import Data.Char (isSpace, isDigit)
11: import Text.Read (readMaybe)
12: -- These are more advanced, used only in 'main'.
13: import Control.Exception (SomeException(..), evaluate, catch)
14: import Control.Monad (when)
15: import System.Exit (exitSuccess)
16: -- The AST type for parsed expression trees
17: data Expr
18:   = Plus Expr Expr
19:   | Minus Expr Expr
20:   | Times Expr Expr
21:   | Divide Expr Expr
22:   | Num Integer
23:   deriving (Eq, Show)
24: -- Possible tokens
25: data Token
26:   = PlusT
27:   | MinusT
28:   | TimesT
29:   | DivideT
30:   | NumT Integer
31:   deriving (Eq, Show)
32: -- Read an input string into a list of tokens.
33: lexTokens :: String -> [Token]
34: lexTokens input = lexNoPrefix (findToken input)
35: -- Drop any non-lexed prefix of the input. This language
36: -- is so simple that we can just use dropWhile.
37: findToken :: String -> String
38: findToken = dropWhile isSpace
39: -- Lex an input string, assuming that the first thing
40: -- in the string (if anything) is a token (as opposed to
41: -- whitespace).
42: lexNoPrefix :: String -> [Token]
43: lexNoPrefix [] = []
44: lexNoPrefix (c:ss) = token : lexTokens rest
45:   where
46:     (token, rest) = lex1 c ss
47: -- Given the first character and the rest of the input string,
48: -- lex one token, returning the remainder of the input string.
49: lex1 :: Char -> String -> (Token, String)
50: lex1 c ss
51:   | isDigit c
52:   , (more_digs, rest) <- span isDigit ss
53:   , Just n <- readMaybe (c:more_digs)
54:   = (NumT n, rest)
55:   | c == '+'
56:   , (rest, rest) = lex1 ' ' ss
57:   , Just n <- readMaybe (c:more_digs)
58:   = (NumT n, rest)
59:   | c == '-'
60:   , (rest, rest) = lex1 ' ' ss
61:   , Just n <- readMaybe (c:more_digs)
62:   = (NumT n, rest)
63:   | c == '*'
64:   , (rest, rest) = lex1 ' ' ss
65:   , Just n <- readMaybe (c:more_digs)
66:   = (NumT n, rest)
67:   | c == '/'
68:   , (rest, rest) = lex1 ' ' ss
69:   , Just n <- readMaybe (c:more_digs)
70:   = (NumT n, rest)
71: lex1 ' ' ss = (PlusT, ss)
72: lex1 ' ' ss = (MinusT, ss)
73: lex1 ' ' ss = (TimesT, ss)
74: lex1 ' ' ss = (DivideT, ss)
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: -- primary user interaction commands
101: putStrLn ""
102: putStrLn "Enter a prefix expression:"
103: expr_string <- getLine
104: expr_string <- getLine
105: -- allow users to quit
106: when (expr_string == "quit")
107: exitSuccess
108: -- This code runs evalString in a way that, if evalString calls 'error',
109: -- the program will not immediately abort. The Haskell features used here
110: -- are beyond the scope of CS245. The curious may enjoy looking these
111: -- functions up online.
112: catch (do value <- evaluate (evalString expr_string)
113:        print value)
114:        (\ (SomeException e) -> print e)
115:        main
116: -- And do it again.
117: main