Expr.hs

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