Homework 6

This week, we'll implement a recursive descent parser for our new language, Boba 1.0.

Boba 1.0 will support the following grammar:

<expr> -> OPENPAREN OPERATOR <operands> CLOSEPAREN |  POSNUMBER

<operands> ->  <expr> [<operands>]

The lexical structure of Boba 1.0 is defined by the following tokens:

OPENPAREN: "("

CLOSEPAREN: ")"

OPERATOR :"+" , "-", "\*", "/"

POSNUMBER : this token represents any positive number (including 0).  The number must start and end with a digit (0 through 9), and may contain a decimal point.  Numbers are stored as double precision floating point numbers (Double).

A starter file, [boba1.hs](https://sjsu.instructure.com/courses/1416874/files/62622331?wrap=1)[download](https://sjsu.instructure.com/courses/1416874/files/62622331/download?download_frd=1)is provided for your convenience and includes the relevant type definitions.

The assignment includes 3 parts.

### **Step 1**

Implement a scanner for Boba 1.0 by defining the function scan:

scan :: String -> [Token]

Here are some test cases.  Please feel free to add your own.

> scan "(+ 4.2 (/ 30 5 02) (\* 0.1 (-7)))"

**[OpenParen,Operator '+',PosNum 4.2,OpenParen,Operator '/',PosNum 30.0,PosNum 5.0,PosNum 2.0,CloseParen,OpenParen,Operator '\*',PosNum 0.1,OpenParen,Operator '-',PosNum 7.0,CloseParen,CloseParen,CloseParen]**

scan "(+\*-+4.2()/)"

**[OpenParen,Operator '+',Operator '\*',Operator '-',Operator '+',PosNum 4.2,OpenParen,CloseParen,Operator '/',CloseParen]**

> scan "(+ 10 x)"

**[OpenParen,Operator '+',PosNum 10.0\*\*\* Exception: Lexical Error - invalid character: x**

**CallStack (from HasCallStack):**

**error, called at...**

HINTS:

1. Use the Haskell built-in function lex to split a string into lexemes. Look at the first character of the lexeme to determine the token. Note that lex will keep multiple operators into a single lexeme if they are not separated by space.  You need to add some processing to split them into multiple tokens.
2. Use the *read* function to convert the string representing a number to a double.  Here is an example:

> read "5.9" :: Double

**5.9**

### **Step 2**

Implement a parser for the language by defining the following functions :

parse :: [Token] -> ExpTree

expr :: [Token] -> (ExpTree, [Token])

operands :: [Token] -> ([ExpTree], [Token])

stringToTree:: String -> ExpTree  
stringToTree = parse.scan -- for testing convenience

Here are some test cases.  Please feel free to add your own.

> stringToTree  "(+ 4 (\* 3 5 2) 0.1 (-5.6) 2.3)"

**OpNode '+' [NumNode 4.0,OpNode '\*' [NumNode 3.0,NumNode 5.0,NumNode 2.0],NumNode 0.1,OpNode '-' [NumNode 5.6],NumNode 2.3]**

stringToTree "(+ 4 (\* 3 5 2) 0.1 5.6 (- 5.3 1 2) (/ 10 2))"

**OpNode '+' [NumNode 4.0,OpNode '\*' [NumNode 3.0,NumNode 5.0,NumNode 2.0],NumNode 0.1,NumNode 5.6,OpNode '-' [NumNode 5.3,NumNode 1.0,NumNode 2.0],OpNode '/' [NumNode 10.0,NumNode 2.0]]**

> stringToTree "(+ 4 (\*) 3 5 2 0.1 5.6 2.3)"

**\*\*\* Exception: Parse Error: invalid expression [CloseParen,PosNum 3.0,PosNum 5.0,PosNum 2.0,PosNum 0.1,PosNum 5.6,PosNum 2.3,CloseParen]**

> stringToTree "(+ 4 (3 5))"

**\*\*\* Exception: Parse Error: invalid expression [OpenParen,PosNum 3.0,PosNum 5.0,CloseParen,CloseParen]**

> stringToTree "(+ 4 1) 8)"

**\*\*\* Exception: Parse Error - extra tokens: [PosNum 8.0,CloseParen]**

### **Step 3**

Implement an interpreter for the language by defining the following functions:

eval ::  ExpTree -> Double

interpret :: String -> Double  
interpret = eval.parse.scan

Here are some test cases.  Please feel free to add your own.

> interpret "(+ 4 (\* 3 (- 5 2)) 0.1 5.6 (- 5.3 1 2) (/ 10 (-2)))"

**16.0**

interpret "(+ (\* (+ 1 (/ 10 (- 4 2)) 2) 2 3) 1)"

**49.0**

> interpret "(- 4)"

**-4.0**

> interpret "(/ 2)"

**0.5**

> interpret "(/ 100 10  2)"

**5.0**

> interpret "(\*2)"

**2.0**

HINT:   Consider using map and one of the fold functions here.

Start early, ask questions and have fun!

