# **Homework 7**

This week, we'll implement an interpreter for Boba 2.0.

Boba 2.0 will support the following grammar:

<expr> -> OPENPAREN OPERATOR <operands> CLOSEPAREN

               |  POSNUMBER

               | OPENPAREN LET  IDENTIFIER <expr>  <expr> CLOSEPAREN

               | <application>

               | IDENTIFIER

<operands> ->  <expr> [<operands>]

<function> -> OPENPAREN LAMBDA OPENPAREN IDENTIFIER CLOSEPAREN <expr> CLOSEPAREN

<application> ->  OPENPAREN <function> <expression> CLOSEPAREN

The operators include +, -, \*, and /  and numbers are stored as double precision floating point numbers (Double).

The language recognizes two keywords:  "let" and "lambda".

Identifiers consist of any name that is not a keyword, and that starts with a letter or an underscore and contains letters, underscores, digits or a single quote.

A starter file, [boba2.hs](https://sjsu.instructure.com/courses/1416874/files/62850148?wrap=1)[download](https://sjsu.instructure.com/courses/1416874/files/62850148/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 the language by defining the function *scan*:

scan :: String -> [Token]

Here are some test examples:

Boba2> scan "(let grade\_1 (+ 80 1) (\* grade\_1 1.23))"

**[OpenParen,Let,Identifier "grade\_1",OpenParen,Operator '+',PosNum 80.0,PosNum 1.0,CloseParen,OpenParen,Operator '\*',Identifier "grade\_1",PosNum 1.23,CloseParen,CloseParen]**

Boba2> scan "((lambda (grade3) (+ grade3 5)) (/ (+ 100 80) 2))"

**[OpenParen,OpenParen,Lambda,OpenParen,Identifier "grade3",CloseParen,OpenParen,Operator '+',Identifier "grade3",PosNum 5.0,CloseParen,CloseParen,OpenParen,Operator '/',OpenParen,Operator '+',PosNum 100.0,PosNum 80.0,CloseParen,PosNum 2.0,CloseParen,CloseParen]**

Boba2> scan "(- 1 lambda0)"

**[OpenParen,Operator '-',PosNum 1.0,Identifier "lambda0",CloseParen]**

Boba2> scan "(+ 10 x)"

**[OpenParen,Operator '+',PosNum 10.0,Identifier "x",CloseParen]**

\*Boba2> scan "(+ \_x)"

**[OpenParen,Operator '+',Identifier "\_x",CloseParen]**

\*Boba2> scan "(-x)"

**[OpenParen,Operator '-',Identifier "x",CloseParen]**

Boba2> scan "(+ 2 a$)"

**[OpenParen,Operator '+',PosNum 2.0,Identifier "a"\*\*\* Exception: Lexical Error - invalid character: $**

**CallStack (from HasCallStack):**

**error, called at...**

Step 2

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

parse :: [Token] -> ExpTree

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

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

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

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

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

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

Boba2> stringToTree "(let grade (+ 80 1) (\* grade 1.2))"

**LetNode "grade" (OpNode '+' [NumNode 80.0,NumNode 1.0]) (OpNode '\*' [IdentNode "grade",NumNode 1.2])**

Boba2> stringToTree "((lambda (x) (+ x (\* 2  3))) 5)"

**Application (FunctionNode "x" (OpNode '+' [IdentNode "x",OpNode '\*' [NumNode 2.0,NumNode 3.0]])) (NumNode 5.0)**

Boba2> stringToTree   "(let first 1 (let second 2 (+ first second)))"

**LetNode "first" (NumNode 1.0) (LetNode "second" (NumNode 2.0) (OpNode '+' [IdentNode "first",IdentNode "second"]))**

Boba2> stringToTree  "(let y 6 ((lambda (x) (+ x (\* y  3))) 5))"

**LetNode "y" (NumNode 6.0) (Application (FunctionNode "x" (OpNode '+' [IdentNode "x",OpNode '\*' [IdentNode "y",NumNode 3.0]])) (NumNode 5.0))**

Boba2> stringToTree "(+ 4 1) grade)"

**\*\*\* Exception: Parse Error - extra tokens: [Identifier "grade",CloseParen]**

**CallStack ...**

Boba2> stringToTree "(let grade 5 +)"

**\*\*\* Exception: Parse Error: invalid function application: [Operator '+',CloseParen]**

Boba2> stringToTree "((lambda (x) 5"

**\*\*\* Exception: Invalid function: [PosNum 5.0]**

Step 3

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

eval :: [(String, Double)] -> ExpTree -> Double

eval0 :: ExpTree -> Double

interpret :: String -> Double  
interpret = eval0.build.scan

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

Boba2> interpret "(let x (/ 2)  (\* 1.5 x))"

**0.75**

Boba2> interpret "(let x 2 (-x))"

**-2.0**

\*Boba2> interpret "(let x 2 (-(+x 1)))"

**-3.0**

Boba2> interpret "((lambda (x) (+ 1 x (\* 2  3))) 5)"

**12.0**

Boba2>interpret  "(let \_first 5 (let second 50 (-  second \_first)))"

**45.0**

Boba2>  interpret  "(let y 5 ((lambda (x) (+ x (\* y  3))) 1.5))"

**16.5**

Boba2> interpret  "(let y 6 ((lambda (x) (let y 2 (+ x (\* y  3)))) 5))"

**11.0**

Boba2>  interpret  "(let y 2 ((lambda (y) (+ 1 (\* y  3))) 5))"

**16.0**

Boba2> interpret "((lambda (x) (+ x ((lambda (y) (\* x y)) 3) (\* 2  3))) 5)"

**26.0**

Boba2> interpret "((lambda (x) (+ x ((lambda (x) (\* x x)) 3) (\* 2  3))) 5)"

**20.0**

\*Boba2> interpret "(+ x 1)"

**\*\*\* Exception: Undefined Identifier: x**

**Start early**, ask questions and have fun!

