CSE210A HW1: ARITH Report

rssato

January 2021

1 Implementation

The data structure for the AST of ARITH was a tree. The tree consists of two types of nodes: BinOp and Num. The BinOp has an operation and a left and right child. The Num nodes only have integer values.

The input string is parsed into TOKENS. These tokens are objects that contain a type and value. The tokens in my implementation are INTEGER, PLUS, MINUS, MUL, LPAREN, RPAREN, and EOF. The tokens are appended to a list. This is all done in the Lexer class by the tokenize() function.

The list from Lexer is used to build the AST in the Parser. The parser breaks down the tokens into expressions, factors and terms. Expressions are terms plus/minus terms. Terms are factors multiply/divide factors (divide not implemented). Factors are integers or an expression inside parenthesis. This handles the order of operations. Reference: https://ruslanspivak.com/lsbasi-part7/

The interpreter is implemented in the Interpreter class that takes in the root of the AST and evaluates it using the eval() function. This checks the node types and calls the appropratie arithmentic function.

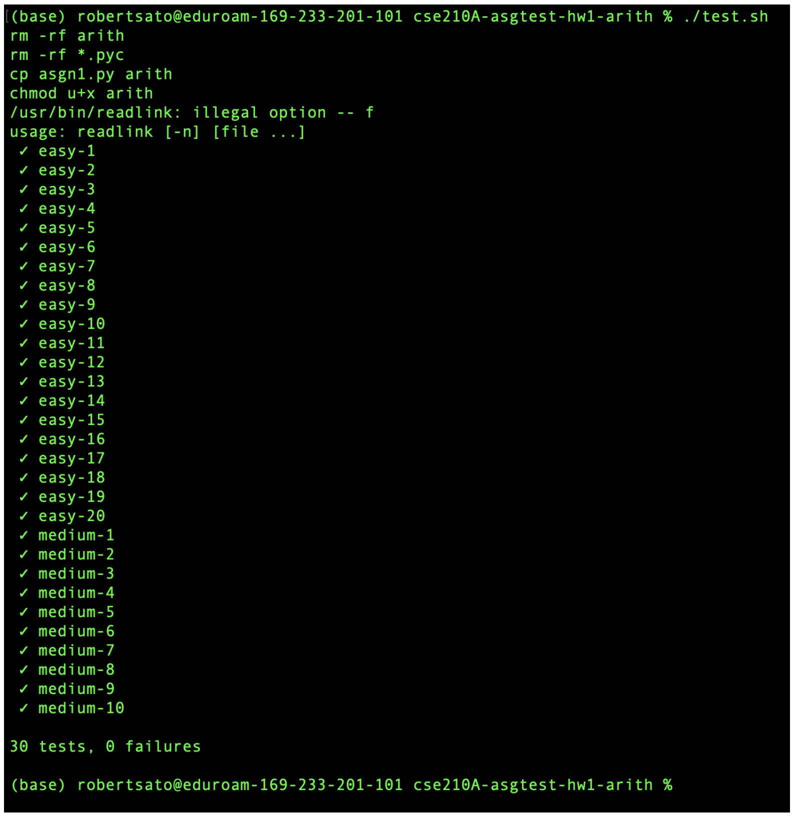
My implementation extends ARITH by adding functionality for subtraction.

2 Test Cases

My ARITH implementation successfully passes all of the test cases as seen in Figure 1.

Figure 2 and 3 shows output for testing that the input equation is properly being converted to tokens

Figure 4 and 5 shows the AST printed in post-order. Therefore, the order of operations can be read top down while the numbers they act on are read bottom up.

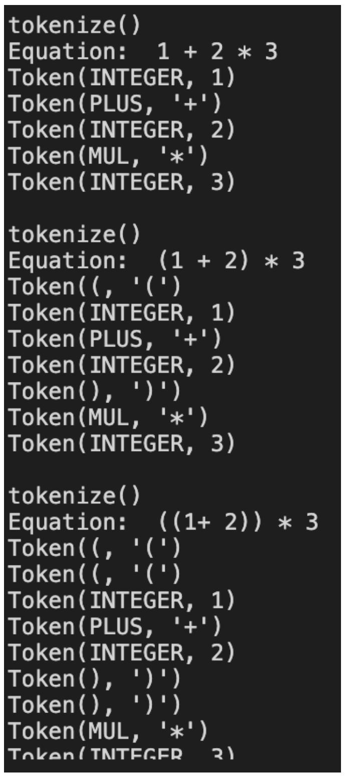


```
(base) robertsato@eduroam-169-233-201-101 cse210A-asgtest-hw1-arith % ./test.sh
rm -rf arith
rm -rf *.pyc
cp asgn1.py arith
chmod u+x arith
/usr/bin/readlink: illegal option -- f
usage: readlink [-n] [file ...]
 ✓ easy-1
 ✓ easy-2
 ✓ easy-3
 ✓ easy-4
 ✓ easy-5
 ✓ easy-6
 ✓ easy-7
 ✓ easy-8
 ✓ easy-9
 ✓ easy-10
 ✓ easy-11
 ✓ easy-12
 ✓ easy-13
 ✓ easy-14
 ✓ easy-15
 ✓ easy-16
 ✓ easy-17
 ✓ easy-18
 ✓ easy-19
 ✓ easy-20
 ✓ medium-1
 ✓ medium-2
 ✓ medium-3
 ✓ medium-4
 ✓ medium-5
 ✓ medium-6
 ✓ medium-7
 ✓ medium-8
 ✓ medium-9
 ✓ medium-10

30 tests, 0 failures

(base) robertsato@eduroam-169-233-201-101 cse210A-asgtest-hw1-arith %
```

Figure 1: ./test.sh

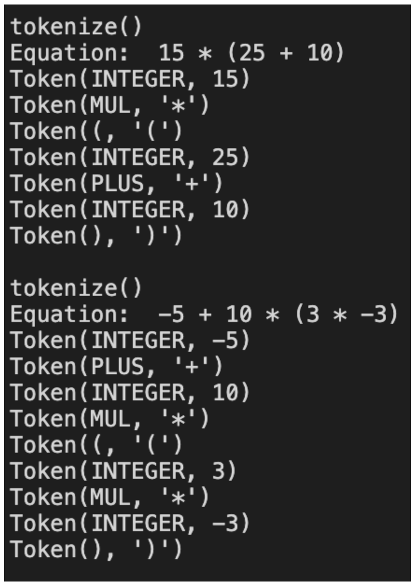


```
tokenize()
Equation:  1 + 2 * 3
Token(INTEGER, 1)
Token(PLUS, '+')
Token(INTEGER, 2)
Token(MUL, '*')
Token(INTEGER, 3)

tokenize()
Equation:  (1 + 2) * 3
Token((, '(')
Token(INTEGER, 1)
Token(PLUS, '+')
Token(INTEGER, 2)
Token(), ')')
Token(MUL, '*')
Token(INTEGER, 3)

tokenize()
Equation:  ((1+ 2)) * 3
Token((, '(')
Token((, '(')
Token(INTEGER, 1)
Token(PLUS, '+')
Token(INTEGER, 2)
Token(), ')')
Token(), ')')
Token(MUL, '*')
Token(INTEGER, 3)
```

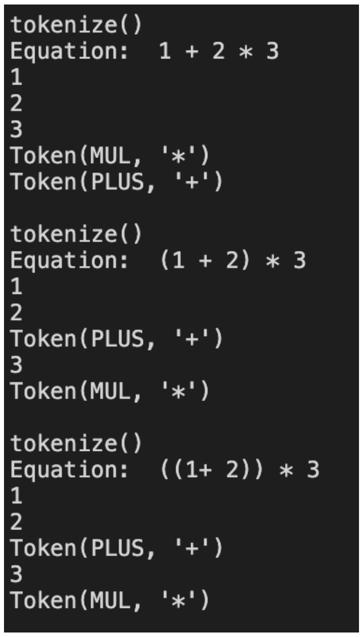
Figure 2: tokenize() generating tokens



```
tokenize()
Equation:  15 * (25 + 10)
Token(INTEGER, 15)
Token(MUL, '*')
Token((, '(')
Token(INTEGER, 25)
Token(PLUS, '+')
Token(INTEGER, 10)
Token(), ')')

tokenize()
Equation:  -5 + 10 * (3 * -3)
Token(INTEGER, -5)
Token(PLUS, '+')
Token(INTEGER, 10)
Token(MUL, '*')
Token((, '(')
Token(INTEGER, 3)
Token(MUL, '*')
Token(INTEGER, -3)
Token(), ')')
```

Figure 3: tokenize() generating tokens

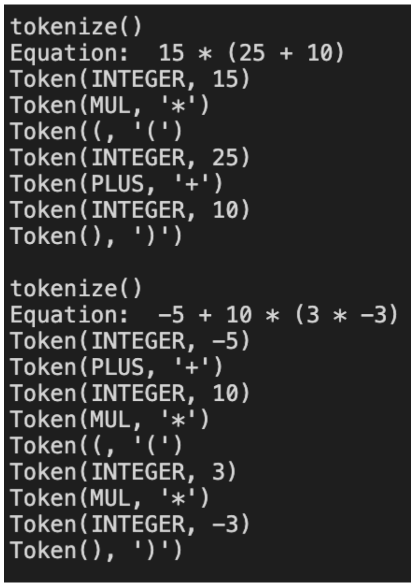


```
tokenize()
Equation:  1 + 2 * 3
1
2
3
Token(MUL, '*')
Token(PLUS, '+')

tokenize()
Equation:  (1 + 2) * 3
1
2
Token(PLUS, '+')
3
Token(MUL, '*')

tokenize()
Equation:  ((1+ 2)) * 3
1
2
Token(PLUS, '+')
3
Token(MUL, '*')
```

Figure 4: post order traversal of the AST



```
tokenize()
Equation:  15 * (25 + 10)
Token(INTEGER, 15)
Token(MUL, '*')
Token((, '(')
Token(INTEGER, 25)
Token(PLUS, '+')
Token(INTEGER, 10)
Token(), ')')

tokenize()
Equation:  -5 + 10 * (3 * -3)
Token(INTEGER, -5)
Token(PLUS, '+')
Token(INTEGER, 10)
Token(MUL, '*')
Token((, '(')
Token(INTEGER, 3)
Token(MUL, '*')
Token(INTEGER, -3)
Token(), ')')
```

Figure 5: tokenize() generating tokens