Assignment B

Team: Andreea Matei & Florin Deleanu

Grammar Design **3**

Test Cases **4**

Test 1 4

Test 2 5

Test 3 8

Grammar Design

The complete structure of the grammar can be seen below:

```
grammar MyGrammar;

myStart :  stat+ EOF;

// rules
stat:   expr      #otherExpr
    |   ID '=' expr  # assign
    |   'print' expr # printExpr
    ;
expr:   expr op=(MUL|DIV) expr #  MulDiv
    |   expr op=(ADD|SUB) expr #  AddSub
    |   '(' expr ')'         # parens
    |   INT          # int
    |   ID           # id
    ;

// tokens
MUL:    '*';
DIV:    '/';
ADD:    '+';
SUB:    '-';
ID:  [_A-Za-z][A-Za-z_!0-9.]* ;
INT     : [0-9]+ ;
WS  : [ \t\r\n]+ -> skip;
```

As shown above, the ‘expr’ element consists of the rules: MulDiv, AddSub, parens, int and id, which are listed in the order of their priority. Therefore, just like in Mathematics and Computer Programming, we must follow some **order of operations** (or **operator precedence**) in order to correctly evaluate a given mathematical expression.

The order of operations, which is used throughout science, technology, is expressed here:

1. exponentiation and root extraction
2. multiplication and division
3. addition and subtraction

This means that if, in a mathematical expression, a subexpression appears between two operators, the operator that is higher in the above list should be applied first.

Since our grammar does not support the exponential and root extraction operations, our first priority should be multiplication and division. One important aspect that should be taken into consideration is that they *BOTH are equally important* and *BOTH are of the same priority*. As a consequence, these two operations ,multiplication and division, must share the same rule and must be the first element that appears in the expr. The same applies in the case of addition and subtraction.

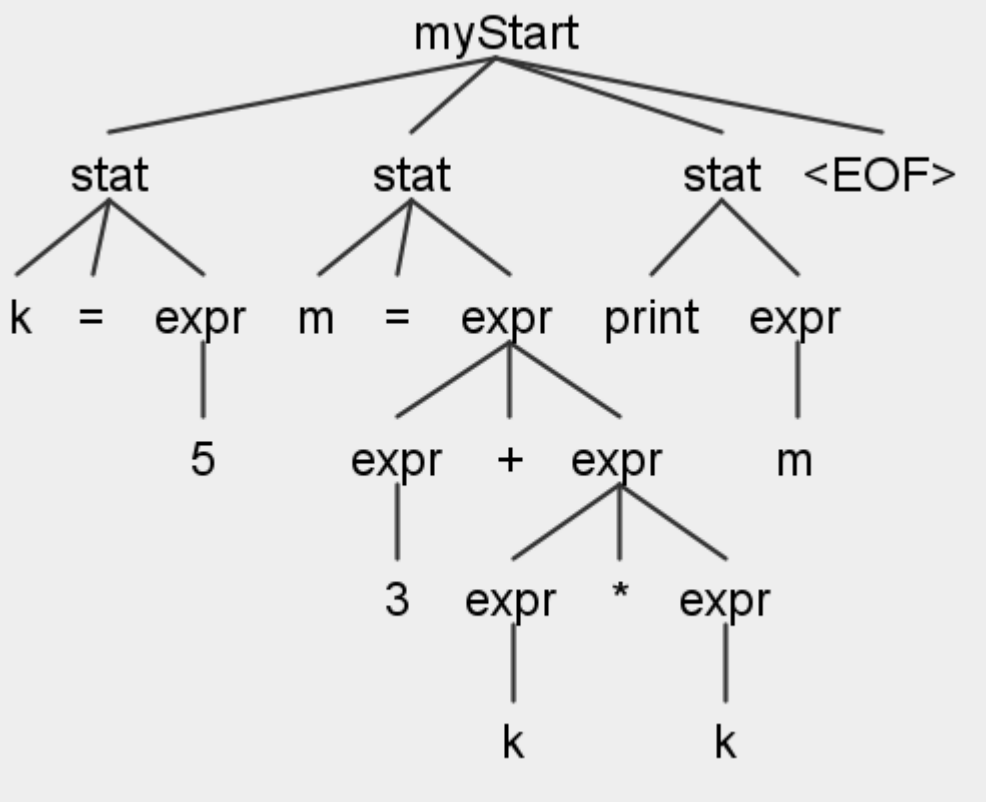
Test Cases

Test 1

Input:

```
k=5
m=3+k*k
print m
```

Parse tree:



Output:

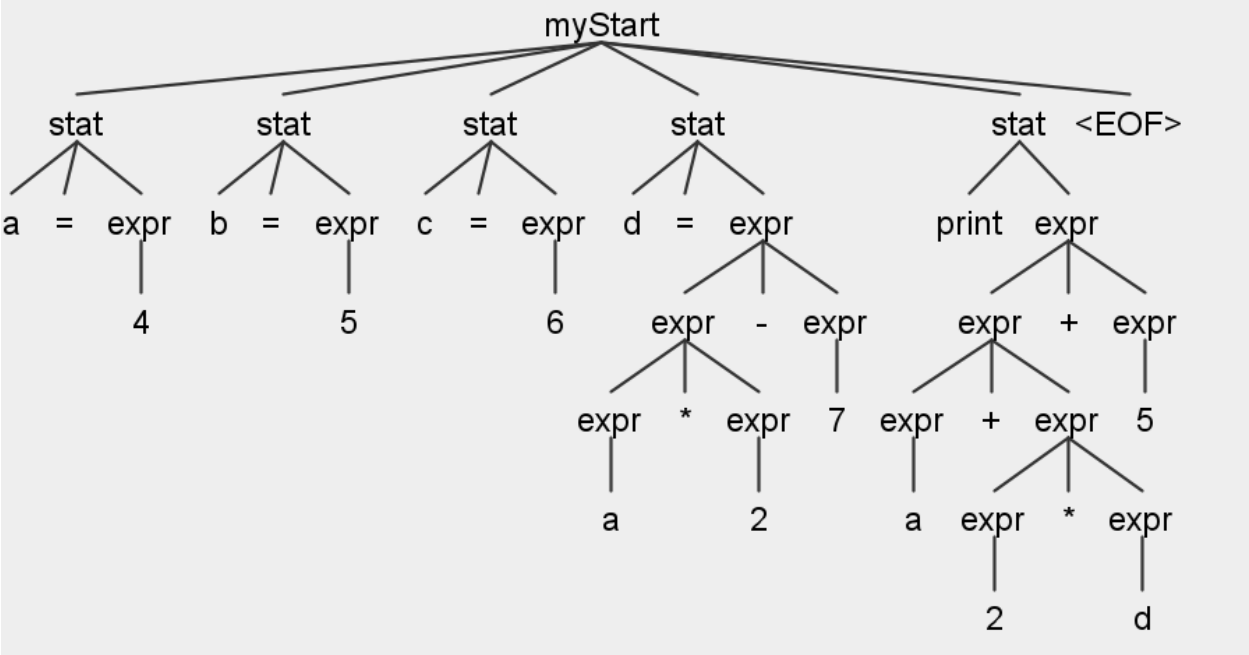
```
G:\TUE courses\data-structures-2\automata\antlr- AssignB>run
enterMyStart()
terminal-node: 'k'
terminal-node: '='
terminal-node: '5'
Added integ to numberStack: 5
memory put: k=5
terminal-node: 'm'
terminal-node: '='
terminal-node: '3'
Added integ to numberStack: 3
terminal-node: '+'
terminal-node: 'k'
Added id to letterstack: k meaning adding 5 to numberstack
terminal-node: '*'
terminal-node: 'k'
Added id to letterstack: k meaning adding 5 to numberstack
multiplied 5 with 5
added 3 with 25
memory put: m=28
terminal-node: 'print'
terminal-node: 'm'
Added id to letterstack: m meaning adding 28 to numberstack
printed m = 28
terminal-node: '<EOF>'
exitMyStart()

G:\TUE courses\data-structures-2\automata\antlr- AssignB>
```

Test 2

Input:

```
a=4
d=a*2-7
print a+2*d+5
```

Parse tree:


myStart
stat
stat
stat
stat
stat
<EOF>
a = expr
b = expr
c = expr
d = expr
print expr
4
5
6
expr - expr
expr + expr
expr * expr
7
expr + expr
5
a
2
a
expr * expr
2
d

```
G:\TUE courses\data-structures-2\automata\antlr- AssignB>run
enterMyStart()
terminal-node: 'a'
terminal-node: '='
terminal-node: '4'
Added integ to numberStack: 4
memory put: a=4
terminal-node: 'b'
terminal-node: '='
terminal-node: '5'
Added integ to numberStack: 5
memory put: b=5
terminal-node: 'c'
terminal-node: '='
terminal-node: '6'
Added integ to numberStack: 6
memory put: c=6
terminal-node: 'd'
terminal-node: '='
terminal-node: 'a'
Added id to letterstack: a meaning adding 4 to numberstack
terminal-node: '*'
terminal-node: '2'
Added integ to numberStack: 2
multiplied 4 with 2
terminal-node: '-'
terminal-node: '7'
Added integ to numberStack: 7
added 8 with 7
memory put: d=1
terminal-node: 'print'
terminal-node: 'a'
Added id to letterstack: a meaning adding 4 to numberstack
terminal-node: '+'
terminal-node: '2'
Added integ to numberStack: 2
terminal-node: '*'
terminal-node: 'd'
Added id to letterstack: d meaning adding 1 to numberstack
multiplied 2 with 1
added 4 with 2
terminal-node: '+'
terminal-node: '5'
Added integ to numberStack: 5
added 6 with 5
printed a+2*d+5 = 11
terminal-node: '<EOF>'
exitMyStart()
```

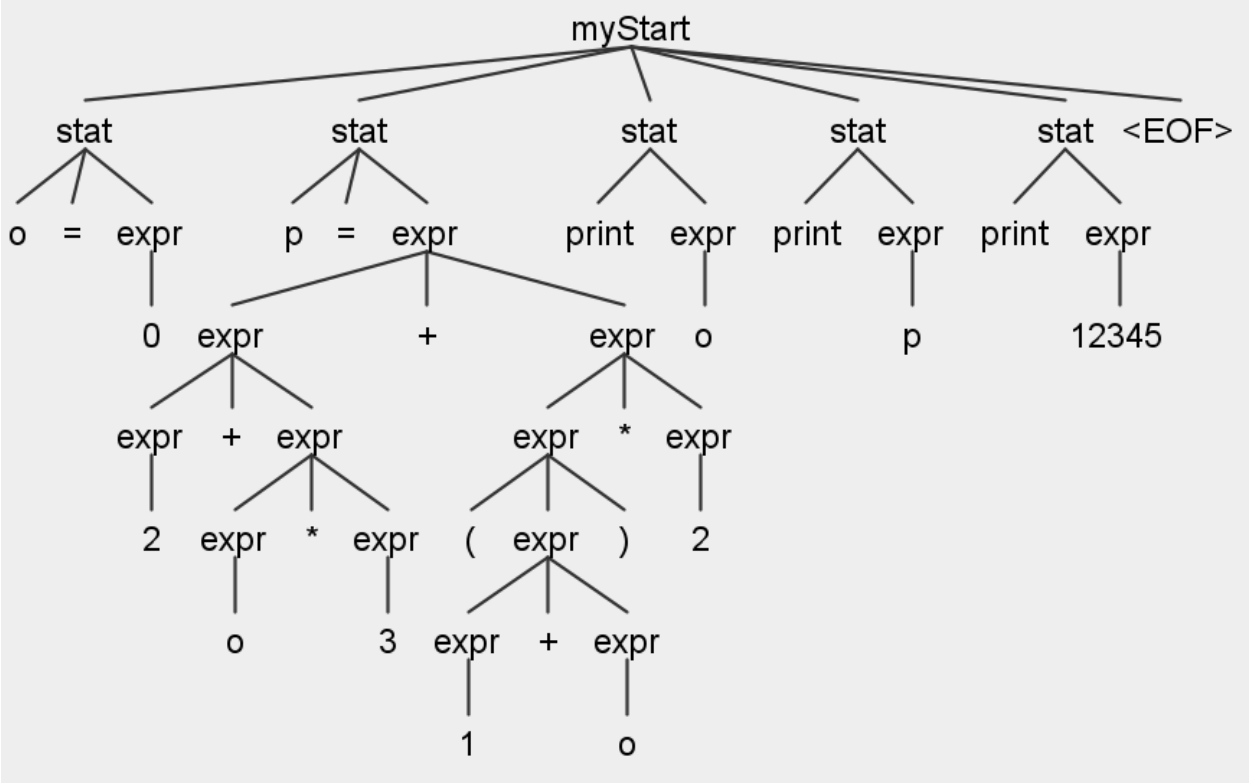
Output:

Test 3

Input:

```
o=0
p=2+o*3+(1+o)*2
print o
print p
print 12345
```

Parse tree:



Output:

```
enterMyStart()
terminal-node: 'o'
terminal-node: '='
terminal-node: '0'
Added integ to numberStack: 0
memory put: o=0
terminal-node: 'p'
terminal-node: '='
terminal-node: '2'
Added integ to numberStack: 2
terminal-node: '+'
terminal-node: 'o'
Added id to letterstack: o meaning adding 0 to numberstack
terminal-node: '*'
terminal-node: '3'
Added integ to numberStack: 3
multiplied 0 with 3
added 2 with 0
terminal-node: '+'
terminal-node: '('
terminal-node: '1'
Added integ to numberStack: 1
terminal-node: '+'
terminal-node: 'o'
Added id to letterstack: o meaning adding 0 to numberstack
added 1 with 0
terminal-node: ')'
terminal-node: '*'
terminal-node: '2'
Added integ to numberStack: 2
multiplied 1 with 2
added 2 with 2
memory put: p=4
terminal-node: 'print'
terminal-node: 'o'
Added id to letterstack: o meaning adding 0 to numberstack
printed o = 0
terminal-node: 'print'
terminal-node: 'p'
Added id to letterstack: p meaning adding 4 to numberstack
printed p = 4
terminal-node: 'print'
terminal-node: '12345'
Added integ to numberStack: 12345
printed 12345 = 12345
terminal-node: '<EOF>'
exitMyStart()
```