Week 1- AUT Assignment

Group 3

Table of contents

Problem description

Assignment 1: expressions

- Write a grammar for numerical expressions (*, /, +, - (and !, ^ if you like))
- Generate an Antlr lexer and parser and check the parse tree with the TestRig (with option '-gui')
- Write your inherited listener to calculate the result of the expression
- Tips for further investigations: exploit the Antlr classes (like: print the vocabulary and the token stream of the lexer, and print all information of a rule's context and a TerminalNode)

Implementation

Below, we have supplied the code for our grammar file so that it can be easily seen the way we have implemented the rules for numerical expressions, for our personal language. While designing the rules, we had to follow the order of operations so that we would not end up with abnormal results. It is also crucial to remember that if we have operations of different order, adding parentheses gives the parser a better “idea” of how it should calculate the expression in the right way.

In the MyListener class, we have implemented the “actual way” of how expressions should be calculated.

e.g.

```
@Override
public void exitAdd(MyGrammarParser.AddContext ctx) {
  int right = numberStack.pop();
  int left = numberStack.pop();
  int result = left + right;
  System.err.println("Added " + left + " with " + right);
  numberStack.push(result);
}

@Override
public void exitSub(MyGrammarParser.SubContext ctx) {
  int right = numberStack.pop();
  int left = numberStack.pop();
  int result = left - right;
  System.err.println("Subtracted " + left + " with " + right);
  numberStack.push(result);
```

```
}
```

Besides telling the compiler what it should do if it encounters addition, subtraction etc., we also had to keep in mind what happens if it has to exit an integer or a statement.
Therefore, for exiting an integer, we had to make sure we had the number to the stack and each time we exit a statement, we have to retrieve the result.
These “small” details made us think about how today’s modern programming languages work behind the scenes.
Also, since we cannot divide by 0, in the “exitDiv” function, we throw an arithmetic exception.

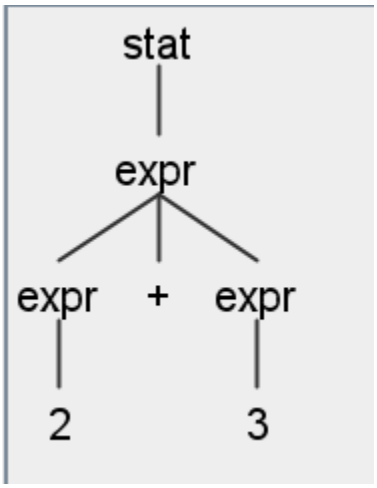
```
            grammar MyGrammar;

// rules
myStart :  stat+ EOF;

// rules
stat:   expr;

expr:  expr op=MUL expr #  Mul
   |   expr op=DIV expr #  Div
   |   expr op=ADD expr #  Add
   |   expr op=SUB expr #  Sub
   |   expr op=POW expr #  Pow
   |   expr op=FACT  #  Fact
   |   PARANL expr PARANR  # parens
   |   INT     # int
   ;

// tokens
MUL:    '*';
DIV:    '/';
ADD:    '+';
SUB:    '-';
POW:    '^';
FACT:   '!';
PARANL: '(';
PARANR: ')';
INT     : [0-9]+ ;
WS  : [ \t\r\n]+ -> skip;
```

Output

Since putting multiple expressions into the input.txt causes the compiler to get confused, we have decided to provide more meaningful test cases within the document.

1. 2 + 3

Parse Tree:

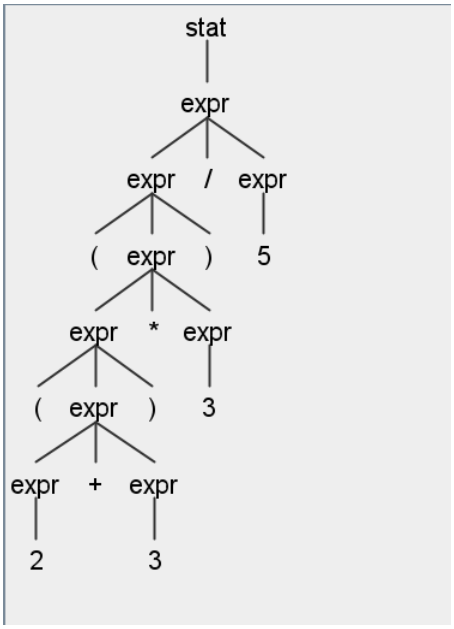


Terminal:

```
enterMyStart()
terminal-node: '2'
terminal-node: '+'
terminal-node: '3'
Added 2 with 3
Final result is: 5
```

2. ((2+3)*3)/5

Parse Tree:

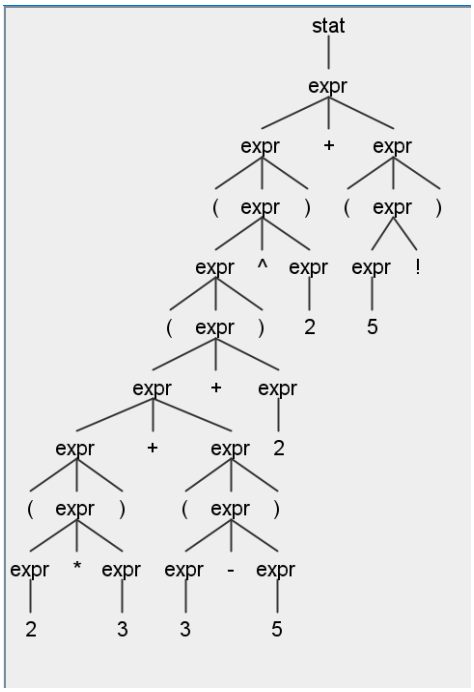


Terminal:

```
terminal-node: '('
terminal-node: '('
terminal-node: '2'
terminal-node: '+'
terminal-node: '3'
Added 2 with 3
terminal-node: ')'
terminal-node: '*'
terminal-node: '3'
Multiplied 5 with 3
terminal-node: ')'
terminal-node: '/'
terminal-node: '5'
Divided 15 with 5
Final result is: 3
```

3. (((2 * 3) + (3 - 5) + 2) ^ 2) + (5!)

Parse Tree:

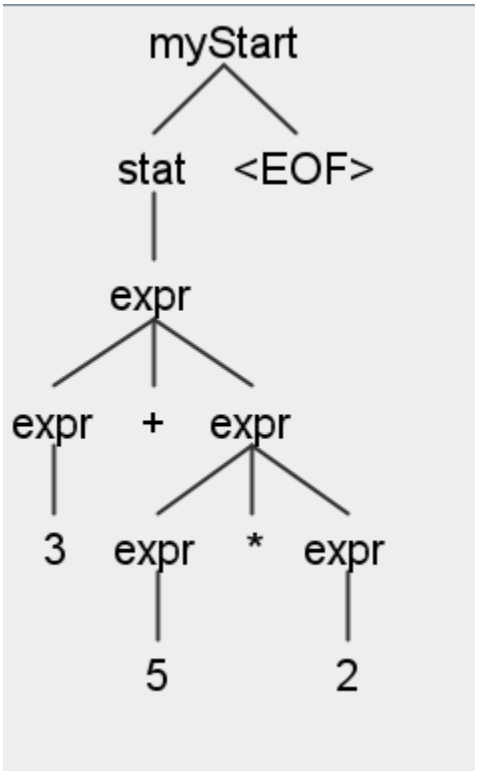


Terminal:

```
terminal-node: ')'
terminal-node: '+'
terminal-node: '('
terminal-node: '3'
terminal-node: '-'
terminal-node: '5'
Subtracted 3 with 5
terminal-node: ')'
Added 6 with -2
terminal-node: '+'
terminal-node: '2'
Added 4 with 2
terminal-node: ')'
terminal-node: '^'
terminal-node: '2'
6 to the power of 2
terminal-node: ')'
terminal-node: '+'
terminal-node: '('
terminal-node: '5'
terminal-node: '!'
Factorial of 5 is 120
terminal-node: ')'
Added 36 with 120
Final result is: 156
terminal-node: '<EOF>'
exitMyStart()
```

4. 3 + 5 * 2

Parse Tree:

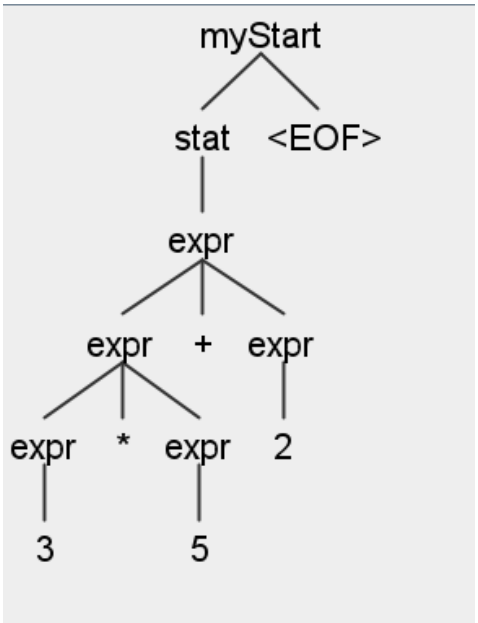


Terminal:

```
terminal-node: '3'
terminal-node: '+'
terminal-node: '5'
terminal-node: '*'
terminal-node: '2'
Multiplied 5 with 2
Added 3 with 10
Final result is: 13
terminal-node: '<EOF>'
exitMyStart()
```

5. 3 * 5 + 2

Parse Tree:

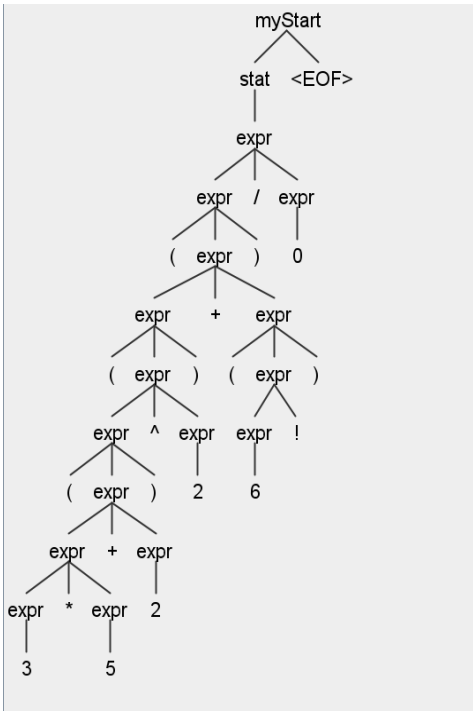


Terminal:

```
enterMyStart()
terminal-node: '3'
terminal-node: '*'
terminal-node: '5'
Multiplied 3 with 5
terminal-node: '+'
terminal-node: '2'
Added 15 with 2
Final result is: 17
terminal-node: '<EOF>'
exitMyStart()
```

6. (((3 * 5 + 2) ^ 2) + (6!)) / 0

Parse Tree:



Terminal:

```
enterMyStart()
terminal-node: '('
terminal-node: '('
terminal-node: '('
terminal-node: '3'
terminal-node: '*'
terminal-node: '5'
Multiplied 3 with 5
terminal-node: '+'
terminal-node: '2'
Factorial of 6 is 720
terminal-node: ')'
Added 289 with 720
terminal-node: ')'
terminal-node: '/'
terminal-node: '0'
Exception in thread "main" java.lang.ArithmeticException: Cannot divide by zero!
```