Programming Assignment 1 - ANTLR4

Programming Languages (SWE3006-42)

Fall, 2022

Instructor:

Sungjae Hwang

- jason.sungjae.hwang@gmail.com
- https://softsec-lab.github.io/

TA:

- Kyeonghwan Min (baksa1849@g.skku.edu)
- Kyuri Choi (gguriskku@gmail.com)

Introduction

- **Deadline : 2022.10.23**

- **You have two days for late submission (~2022.25)**
  - **25%** deduction per day

- **Submit both source code and Report**

Introduction

- **ANTLR(Another Tool for Language Recognition)**
  - A powerful parser generator
  - Parser for reading, processing, executing, or translating structured text or binary files.
  - Widely used to build languages, tools, and frameworks

- **ANTLR**
  - Input: a grammar file (e.g., Hello.g4)
  - Output: parser code in Java (e.g., Hello*.java)

- **You have to get used to Java for this PA1**

Install ANTLR (version 4.9.2) - Java tools

- **ANTLR(www.antlr.org)**
  - https://www.antlr.org/download/antlr-4.9.2-complete.jar

- **Installation JRE/JDK & ANTLR**

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install default-jre
$ sudo apt install default-jdk
$ sudo apt install curl

$ cd /usr/local/lib
$ sudo curl -0 https://www.antlr.org/download/antlr-4.9.2-complete.jar -o
antlr-4.9.2-complete.jar
$ sudo ln -s antlr-4.9.2-complete.jar antlr-complete.jar

$ vi ~/.bashrc
export CLASSPATH=".:/usr/local/lib/antlr-complete.jar:$CLASSPATH"
alias antlr4='java -jar /usr/local/lib/antlr-complete.jar'
alias grun='java org.antlr.v4.gui.TestRig'
```

→ Add 3 lines at the end of ~/.bashrc

```
$ source ~/.bashrc
```

→ Reflect the effect to the current shell

Example Grammar File (*.g4)

```
/* Example grammar for Expr.g4 */
grammar Expr;              // name of grammar

//parser rules – start with lowercase letters
prog: (expr NEWLINE)* ;
expr: expr ('*'|'/') expr
    | expr ('+'|'-') expr
    | INT
    | '(' expr ')' ;

//lexer rules – start with uppercase letters
NEWLINE : [\r\n]+ ;
INT : [0-9]+ ;
WS : [ \t\r\n]+ -> skip;
```

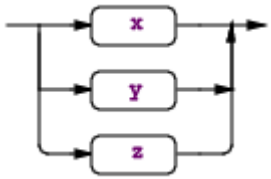
Regular Expressions

- . matches any single character
- * matches zero or more copies of preceding expression
- + matches one or more copies of preceding expression
- ? matches zero or one copy of preceding expression
  - -?[0-9]+ : signed numbers including optional minus sign
- [ ] matches any character within the brackets
  - [Abc1], [A-Z], [A-Za-z], [^123A-Z]
- ^ matches the beginning of line
- $ matches the end of line
- \ escape metacharacter e.g. * matches with *
- | matches either the preceding expression or the following
  - abc|ABC
- ( ) groups a series of regular expression
  - (123)(123)*

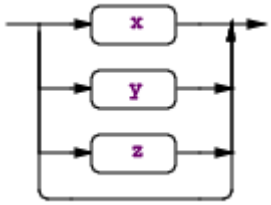
exclude [123A-Z]

Regular Expressions (subrules)

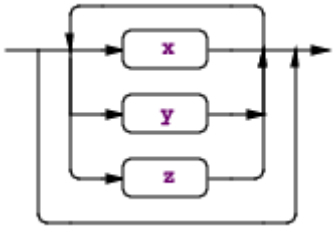
- `(x|y|z)` : match any alternative within the subrule exactly



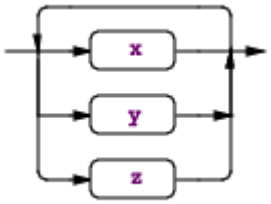
- `(x|y|z)?` : match nothing or any alternative within subrule



- `(x|y|z)*` : match an alternative within subrule zero or more times



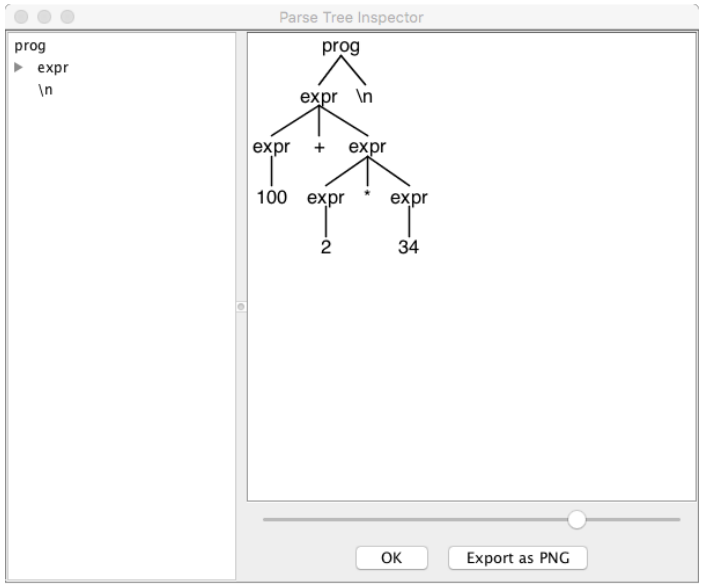
- `(x|y|z)+` : match an alternative within subrule one or more times.



Running ANTLR Parser Generator

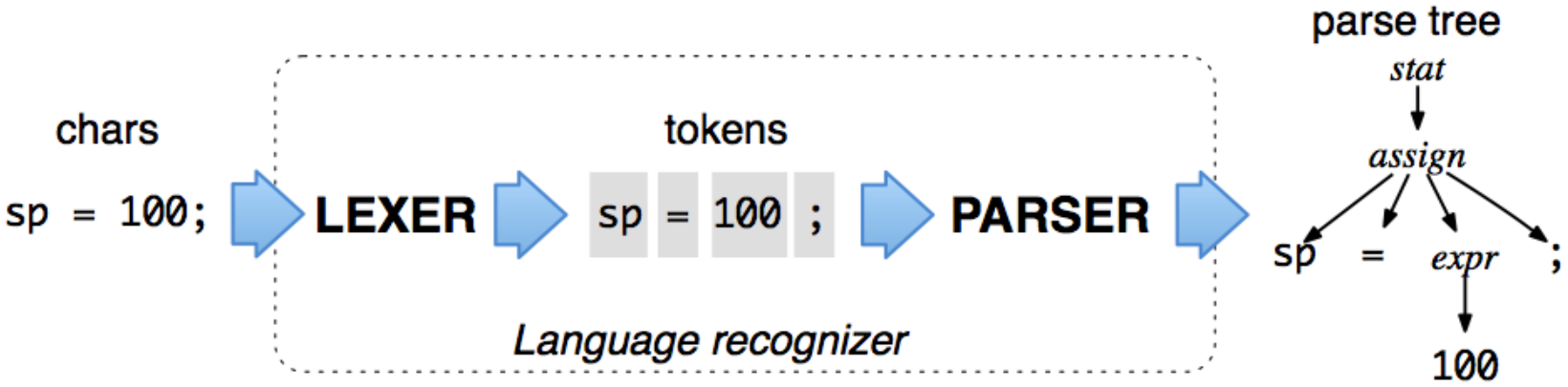
- Writing a grammar file
  - E.g., `Expr.g4` (slide 4)
- Process with ANTLR
  - `$ antlr4 Expr.g4`
- Compile java programs
  - `$ javac Expr*.java`
- Run a generated parse
  - `$ grun Expr prog -gui`
  - `$ grun Expr prog -tree`

```
$ antlr4 Expr.g4
$ javac Expr*.java
$ grun Expr prog -gui
100 + 2*34
^D
```



Parse Tree

- **ANTLR-generated parser builds a data structure**
  - Parse tree (or syntax tree)
  - "Organization of input" according to grammar



Parse Tree Manipulation

- **Now, you have a parse tree**
  - Walk a parse tree with ANTLR tools - Visitor

- **Visitor**
  - Make functions triggered at entering/exit of nodes
  - To generate visitor class, use **-visitor** option for antlr4
    - e.g., **antlr4 -visitor Expr.g4**

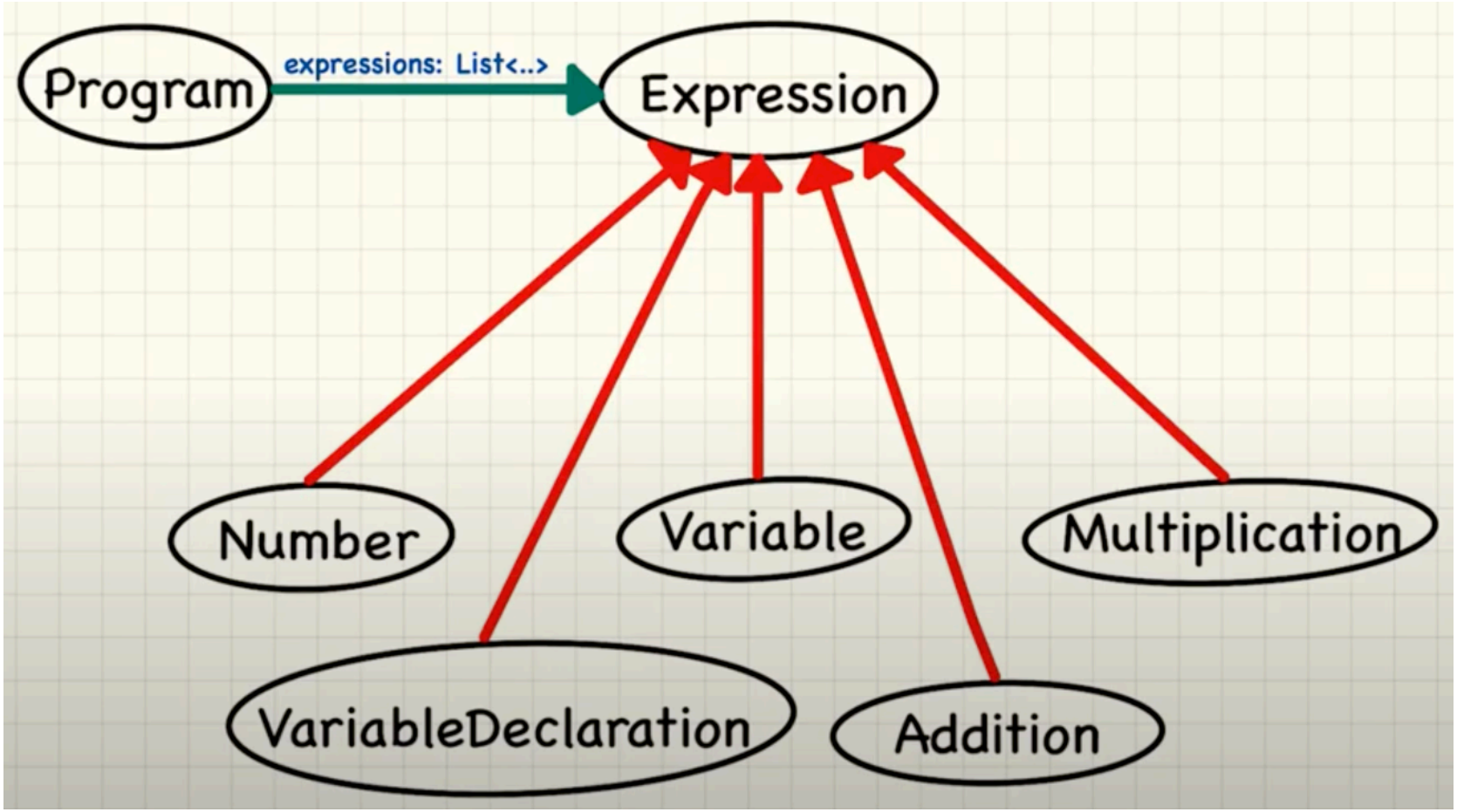
Parse Tree Manipulation

```
// Generated from Expr.g4 by ANTLR 4.9.2
import org.antlr.v4.runtime.tree.AbstractParseTreeVisitor;

/**
 * This class provides an empty implementation of {@link ExprVisitor},
 * which can be extended to create a visitor which only needs to handle a subset
 * of the available methods.
 *
 * @param <T> The return type of the visit operation. Use {@link Void} for
 * operations with no return type.
 */
public class ExprBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements ExprVisitor<T> {
    @Override public T visitProg(ExprParser.ProgContext ctx) { return visitChildren(ctx); }
    @Override public T visitInfixExpr(ExprParser.InfixExprContext ctx) { return visitChildren(ctx); }
    @Override public T visitNumberExpr(ExprParser.NumberExprContext ctx) { return visitChildren(ctx); }
    @Override public T visitParensExpr(ExprParser.ParensExprContext ctx) { return visitChildren(ctx); }
}
```

```
/* Expr.g4  */
grammar Expr;

// parser rules
prog : (expr NEWLINE)*;
expr : expr ('*'|'/') expr # InfixExpr
     | expr ('+'|'-') expr # InfixExpr
     | INT                 # NumberExpr
     | '(' expr ')';       # ParensExpr

// lexer rules
NEWLINE: [\r\n]+;
INT: [0-9]+;
WS: [ \t\r\n]+ -> skip;
```

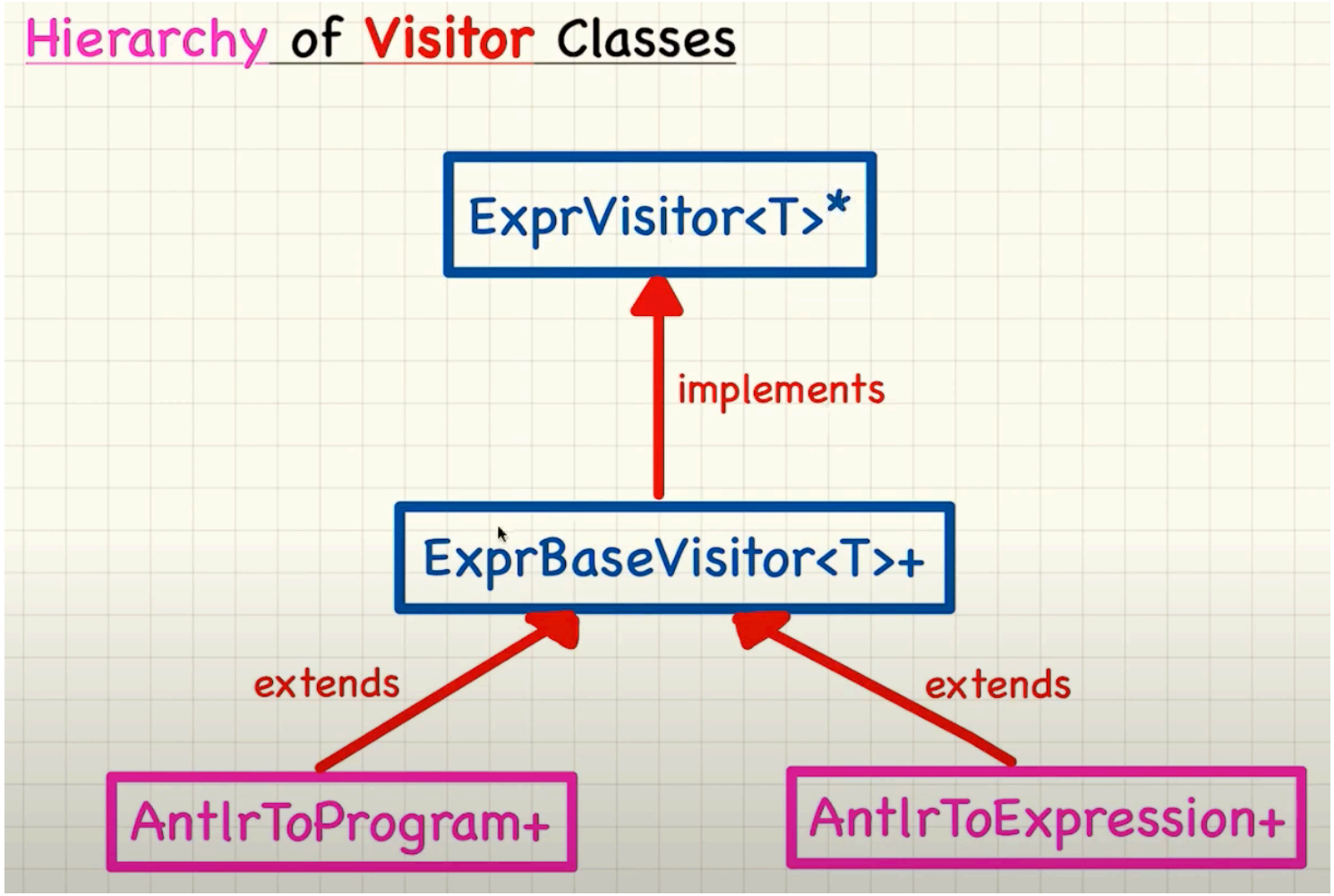
ExprBaseVisitor.java: generated by ANTLR4 along with multiple java files and others

Parse Tree Manipulation



출처: https://www.youtube.com/watch?v=2o9ImGNI1uw&list=PL5dxAmCmjv_4FGYtGzcvBeoS-BobRTJLq&index=3

Parse Tree Manipulation



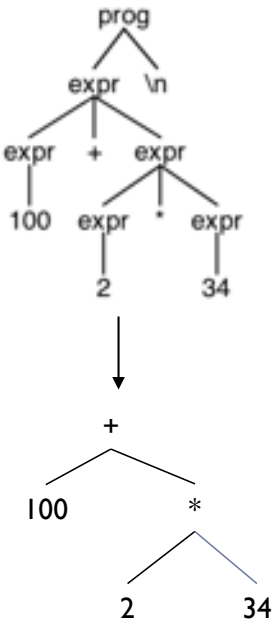
출처: https://www.youtube.com/watch?v=2o9ImGNI1uw&list=PL5dxAmCmjv_4FGYtGzcvBeoS-BobRTJLq&index=3

Parse Tree Manipulation

```
public class BuildAstVisitor extends MathBaseVisitor<T> {

    public T visitProg(MathParser.CompileUnitContext ctx) {
        return visit(ctx.expr());
    }

    public T visitNumberExpr(MathParser.NumberExprContext ctx) {
        return visit(ctx.expr());
    }

    public T visitParensExpr(MathParser.ParensExprContext ctx) {
        return visit(ctx.expr());
    }

    public T visitInfixExpr(MathParser.InfixExprContext ctx) {
        return visit(ctx.expr());
    }
```

PA1

- **Build a Java program using ANTLR Visitor class for Build AST(Abstract Syntax Tree)**
  - Expand grammar to execute following rule
    - Assign number in a variable
      - a = 10;
  - Assign a variable to another variable is not allowed
    - You can only assign integer and real values to variables
  - Call function in java.lang.Math and execute the function.
    - min(5,7) = 5
  - Build AST
    - Antler build parse tree when you execute
    - Convert the parse tree to AST
    - Should use only visitor, not listener



PA1

- **Build a Java program using ANTLR Visitor class for Build AST(Abstract Syntax Tree)**
  - Print AST in terminal
    - This program should print ‘ADD’, ‘MUL’, ‘SUB’, ‘DIV’, ‘ASSIGN’ not ‘+’, ‘*’, ‘-’, ‘/’, ‘=’
  - Calculate the input
    - Calculate the resulting values of expressions
    - The function call expressions will be tested with 4 function in **java.lang.Math**
      - min, max, pow, sort
    - If just assign expression is given as input, **just print 0**
      - Evaluation result of “a=5;” should be 0

PA1

```
yongwoo@fpga2:~/antlr$ java program
3 + 4
ADD
        3.0
        4.0
7.0
yongwoo@fpga2:~/antlr$ java program
3 + 5 * 4
ADD
        3.0
        MUL
                5.0
                4.0
23.0
yongwoo@fpga2:~/antlr$ java program
3 * (3 + 5) * 4
MUL
        MUL
                3.0
                ADD
                        3.0
                        5.0
        4.0
96.0
yongwoo@fpga2:~/antlr$
```

```
yongwoo@fpga2:~/antlr$ java program
sqrt(4)
sqrt
        4.0
2.0
yongwoo@fpga2:~/antlr$
```

```
yongwoo@fpga2:~/antlr$ java program
a = 4
ASSIGN
        a
        4.0
0.0
yongwoo@fpga2:~/antlr$
```

PA1

- **AstNodes.java**
  - Define AST nodes to print
  - The nodes have to defined as class
- **BuildAstVisitor.java**
  - Build AST using ExprBaseVisitor.java
- **AstCall.java**
  - Define methods to print the AST nodes
  - The name of the method should be “Call”
- **Evaluate.java**
  - Define methods to calculate the expression we get as input
  - The name of the method should be “evaluate”

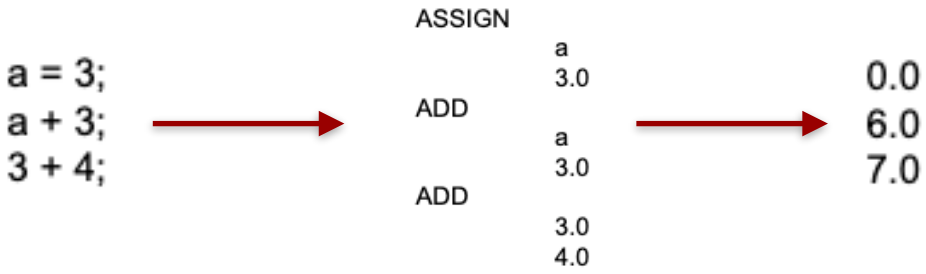
PA1

- **Program.java**
  - Define the main method in the file
    - In the main method,
      - Build parse tree
      - Accept input as command line
      - Call the method as define (call and evaluate)
      - Print out resulting value
      - Calculation should be in double
        - 5/2 = 2.5 not 2.
        - **ctrl + d after you enter input**

PA1

Additional Information

- Input can be multiple lines. Generate one AST
  - e.g.,



- You don't need to consider divide by 0
- Just consider assigning numbers to the variables
  - a=pow(5,5); <- no need to consider this
  - a = b = 5 <- no need to consider this
- Only numbers can be given for the function arguments
- Negative numbers can be used for inputs

PA1

```
wigkx@wigkxerver:~/Desktop/peuon/t$ javac *.java
wigkx@wigkxerver:~/Desktop/peuon/t$ java program
a = 3; a + pow(3,2); a/2+(2*3+5);
ASSIGN
        a
        3.0
ADD
        a
        pow
                3.0
                2.0
ADD
        DIV
                a
                2.0
        ADD
                MUL
                        2.0
                        3.0
                5.0
0.0
12.0
12.5
```

Cheating

- **All assignments should be written individually.**

- **Cheating will make you fail this course**
  - Copying, retyping, outsourcing, submitting copies of others and etc.
  - We will actively check for plagiarism.
  - We have an automated system that computes the similarity between the submitted materials.
  - Disciplinary actions will follow.

Reference

- **The Definitive ANTLR4 Reference - Terence Parr**
- **http://antlr.org > Dev Tools > Resources**
  - Documentation
    - https://github.com/antlr/antlr4/blob/master/doc/index.md
  - Runtime API (look into “Java Runtime” for ANTLR4 APIs)
    - http://www.antlr.org/api/
- **Java util package**
  - https://www.tutorialspoint.com/java/util/index.htm