PA1 - ANTLR Practice

Programming Languages (SWE3006-41)
Fall, 2023

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

TA :
- Kyeonghwan Min

Software Security Lab
SUNGKYUNKWAN UNIV

Introduction

- Deadline : 2023/11/08 (Delay Submission 2023/11/10, 25% deduction per day.)

- Make simple calculator using ANTLR.

- Submit source codes (*.ml) and report.
  - You will not get any points if your source code does not compiles well.
  - Submit "PA1_ANTLR_StudentID.zip" through icampus.
  - The zip file should contains :
    -> ex1.ml, ex2.ml, ex3.ml, ex4.ml, ex5.ml, ex6.ml, ex7.ml

- Please leave the questions in the google sheet.
  https://docs.google.com/spreadsheets/d/19T2K5LmaounOnL-LGaF-nzdu1G_jJXY5nRjSMQr3hVc/edit#gid=959989151
  * Avoid using email or the iCampus message for inquires.

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

  - 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 4.13.1) - Java tools

- ANTLR(www.antlr.org)
  - https://www.antlr.org/download/antlr-4.13.1-complete.jar

- Installation JRE/JDK & ANTLR

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

$ cd /usr/local/lib
$ sudo curl -O https://www.antlr.org/download/antlr-4.13.1-complete.jar -o antlr-4.13.1-complete.jar
$ sudo ln -s antlr-4.13.1-complete.jar antlr-complete.jar

$ cd ~
$ vi .bashrc

# Add 3 lines below at the end of the ~/.bashrc file.
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'

$ source ~/.bashrc
```

Example Grammar File (*.g4)

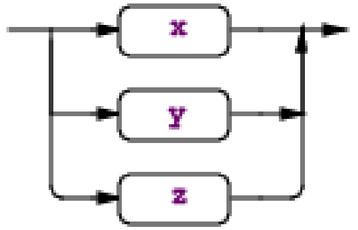
```
/* Example grammar for Expr.g4 */
grammar Expr;              // name of grammar

//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 Expression

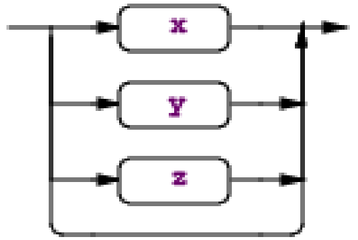
- . matches any single character
- * matches zero or more copies of preceding expression
- + matches one or more copies of preceding expression
- ? matches zero or one copy of preceding expression
  - -?[0-9]+ : signed numbers including optional minus sign
- [ ] matches any character within the brackets
  - [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)*

Regular Expression (subrules)

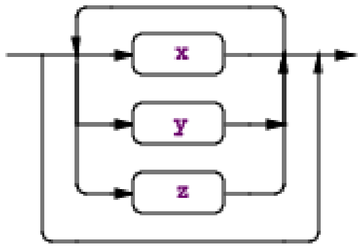
- (x|y|z) : match any alternative
  within the subrule exactly



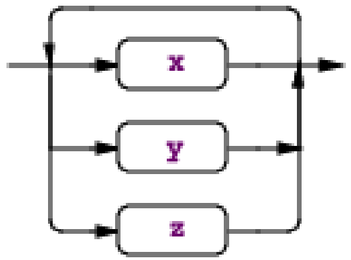
- (x|y|z)? : match nothing or any
  alternative within subrule



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



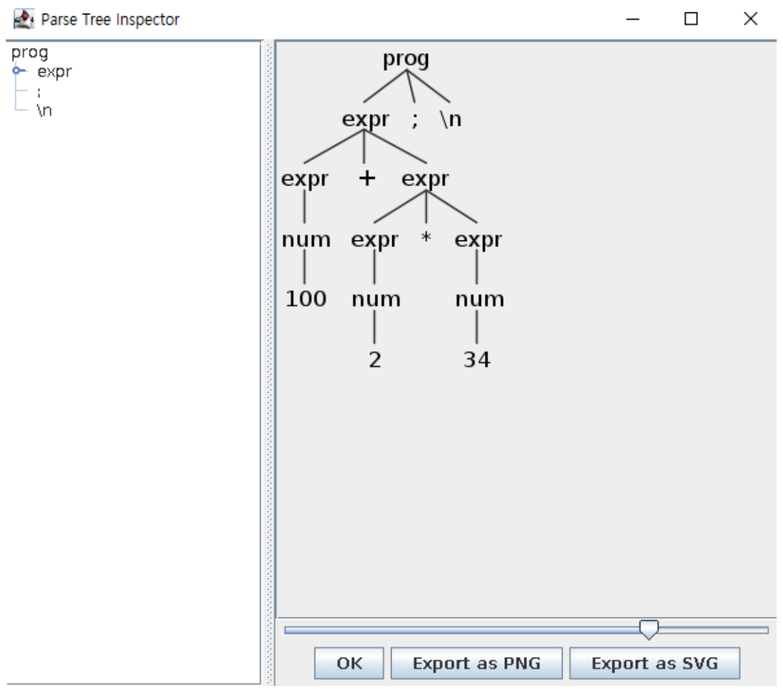
- (x|y|z)+ : match an alternative within
  subrule one or more times.



Running ANTLR Parser Generator

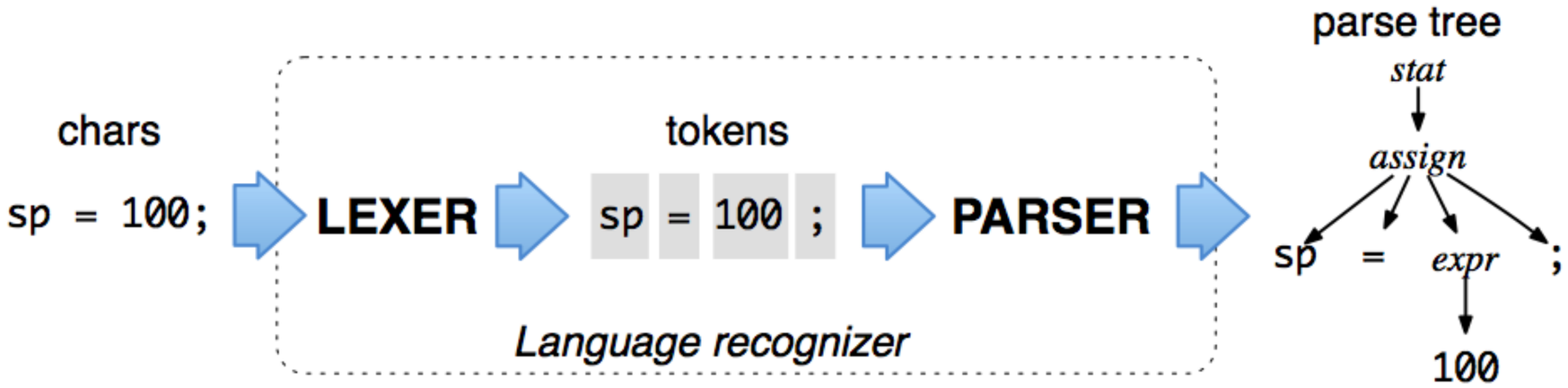
- Writing a grammar file
  - E.g., Expr.g4 (slide 4)
- Process with ANTLR
  - $ antlr4 Expr.g4
- Compile java programs
  - $ javac Expr*.java
- Run a generated parse
  - $ grun Expr prog -gui
  - $ grun Expr prog -tree

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



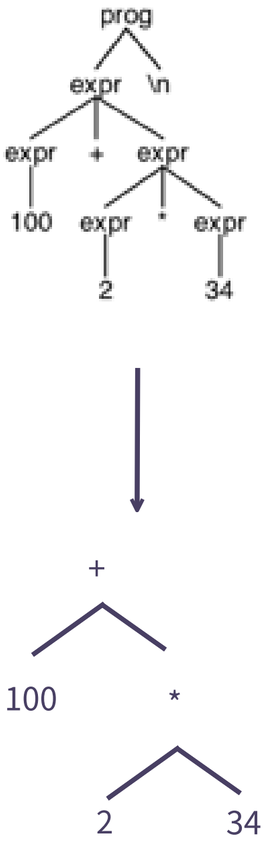
Parse Tree

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



PA1

- Build AST(Abstract Syntax Tree) using ANTLR Visitor class with Java
  - 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
  - Build AST
    - ANTLR builds parse tree when you execute
    - Convert the parse tree to AST
    - Should use only visitor, not listener



PA1

- Build AST(Abstract Syntax Tree) using ANTLR Visitor class with Java
  - Print AST in terminal
    - This program should print ‘ADD’, ‘MUL’, ‘SUB’, ‘DIV’, ‘ASSIGN’ not ‘+’, ‘*’, ‘-’, ‘/’, ‘=’
  - Calculate the input
    - Calculate the resulting values of expressions
    - If just assign expression is given as input, just print 0
      - Evaluation result of “a=5;” should be 0

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

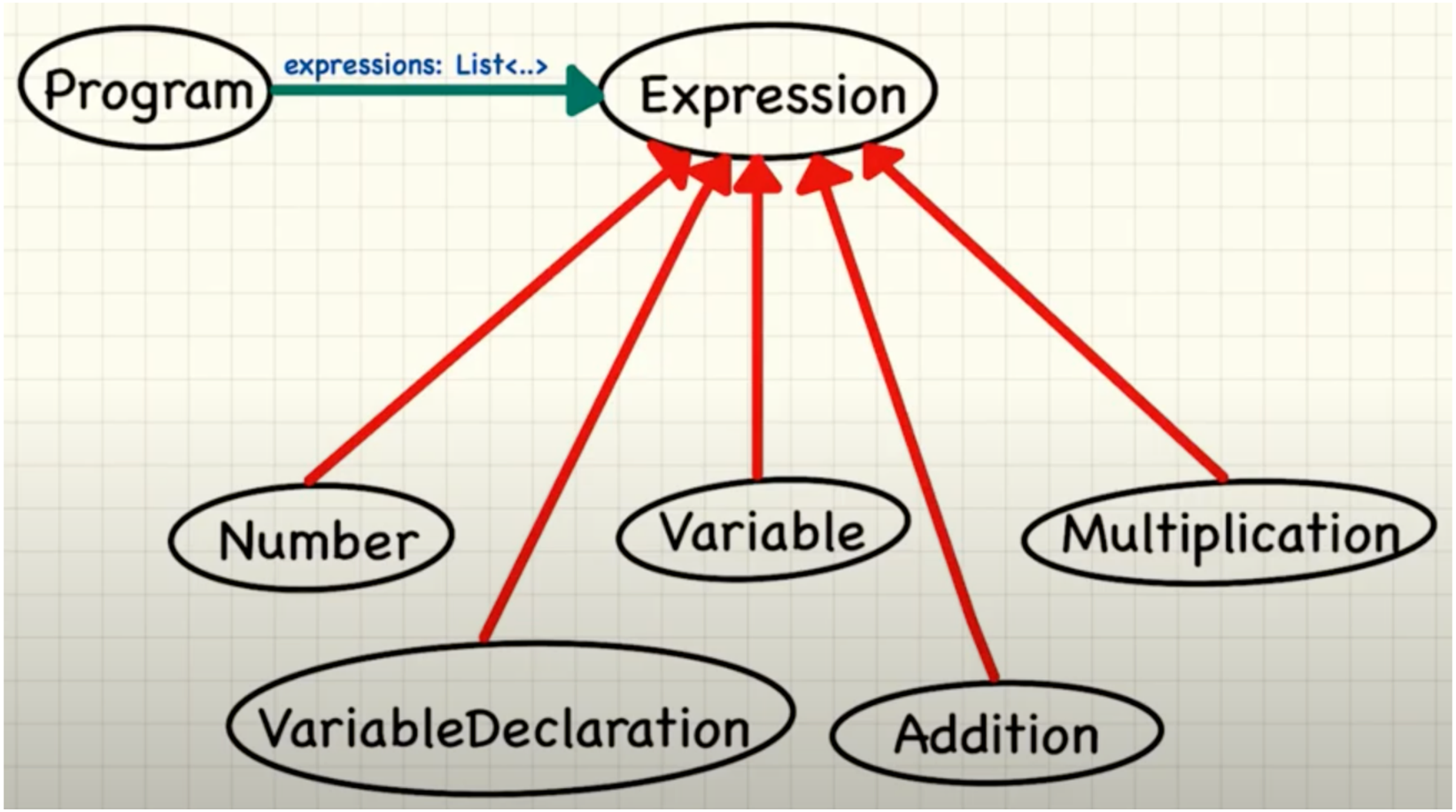
Parse Tree Manipulation

```
// 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;
```

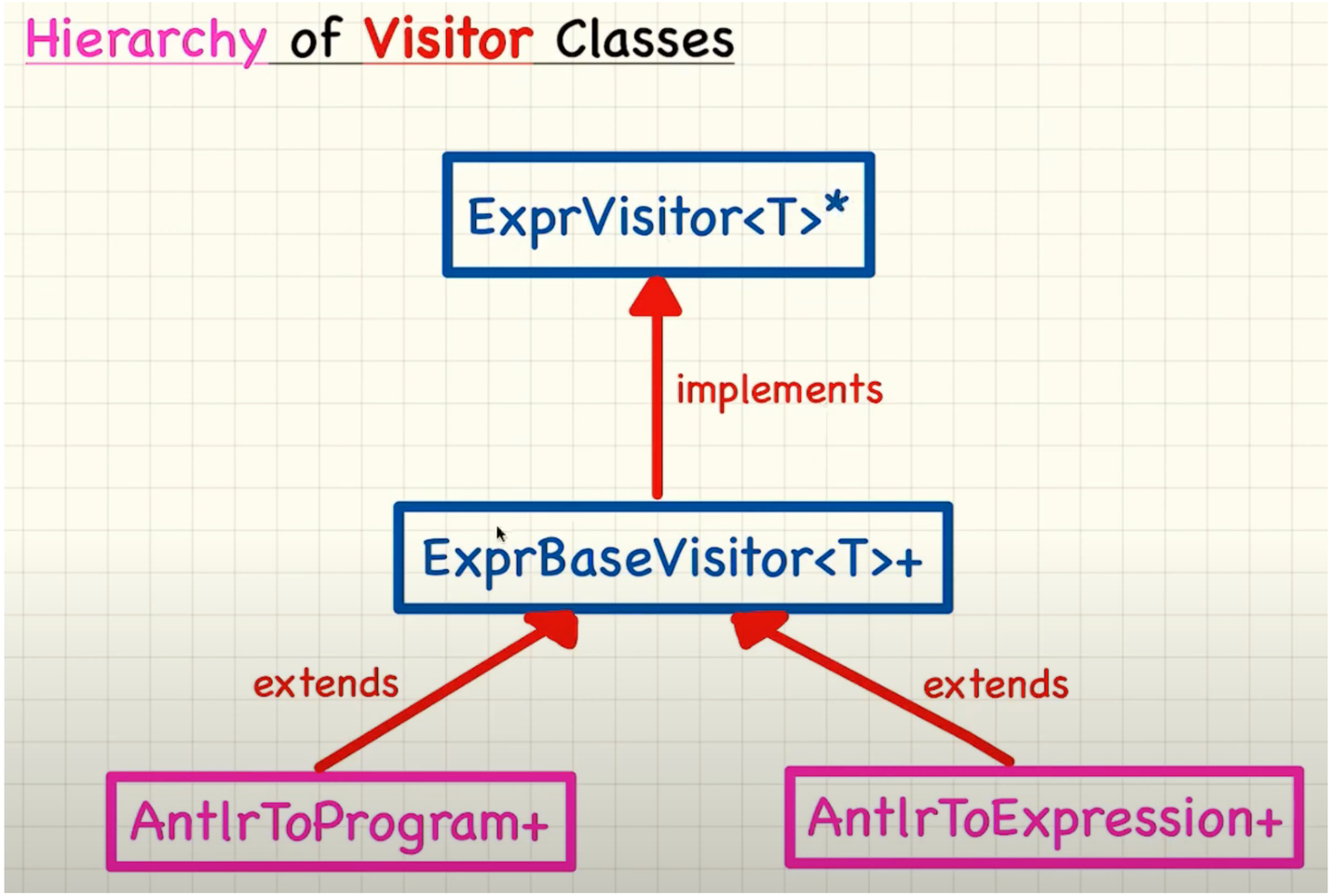
ExprBaseVisitor.java: generated by ANTLR4 along with multiple java files and others

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

```
daily@DESKTOP-AJKPT60:~/2023_Fall_PL/antlr$ java program
3 + 4;
ADD
        3.0
        4.0
7.0
daily@DESKTOP-AJKPT60:~/2023_Fall_PL/antlr$ java program
3 + 5 * 4;
ADD
        3.0
        MUL
                5.0
                4.0
23.0
daily@DESKTOP-AJKPT60:~/2023_Fall_PL/antlr$ java program
3 * (3 + 5) * 4;
MUL
        MUL
                3.0
                ADD
                        3.0
                        5.0
        4.0
96.0
```

```
daily@DESKTOP-AJKPT60:~/2023_Fall_PL/antlr$ java program
a = 4;
ASSIGN
        a
        4.0
0.0
```

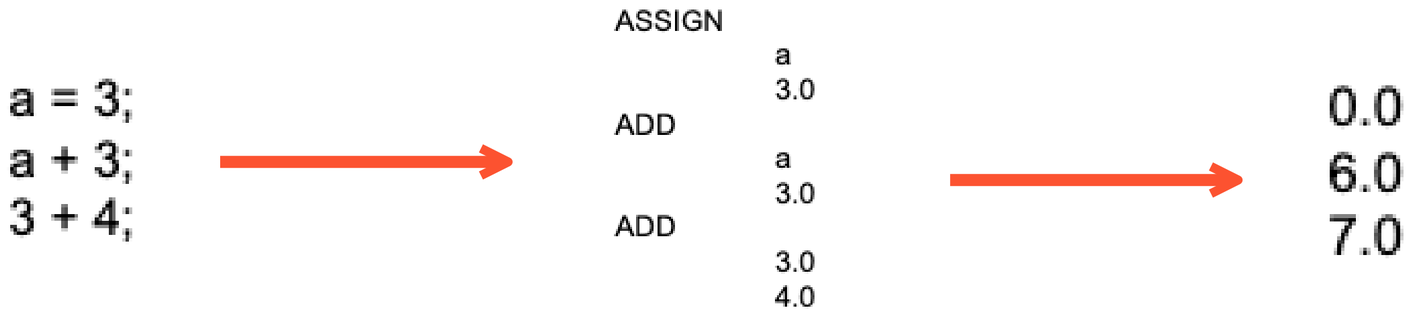
Note) Like figures in this slide,
Each statement would be ended
with semicolon in our assignment.

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”

- 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
  - The source code of Program.java is fully provided.

- 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 = b = 5 <- no need to consider this
  - Only numbers can be given for the function arguments
  - Negative numbers can be used for inputs

- Additional Information
  - You are allowed to add at most 1 custom java file.
  - You are allowed to modify Program.java file.

PA1

```
daily@DESKTOP-AJKPT60:~/2023_Fall_PL/antlr$ java program
a = 3; a + 3 * 3; a / 2 + (2 * 3 + 5);
ASSIGN
        a
        3.0
ADD
        a
        MUL
                3.0
                3.0
ADD
        DIV
                a
                2.0
        ADD
                MUL
                        2.0
                        3.0
                5.0
0.0
12.0
12.5
```

Grading

- Test cases (40pt) + report (10pt)
- 6 publicly available test cases (slide 17, 22)
- Your output should not display any redundant message
  - Only print AST and evaluation
  - Otherwise, it is considered wrong output
- Report
  - Briefly introduce your grammar and code.
  - You don't have to explain your code line by line.
  - You have to submit PDF file format.

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 and AI-generated code.
  - 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