SUNG KYUN KWAN UNIVERSITY

ARCS

ANTLR 4

tutorial + PA#1

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)

Download (ANTLR version 4.5.2)

- ANTLR (www.antlr.org)
  - Download link
  - http://www.antlr.org/download/antlr-4.5.2-complete.jar

- Installation

```
$ cd /usr/local/lib
$ sudo curl -0 http://www.antlr.org/download/antlr-4.5.2-complete.jar

$ export CLASSPATH=".:/usr/local/lib/antlr-4.5.2-
complete.jar:$CLASSPATH"
$ alias antlr4='java -jar /usr/local/lib/antlr-4.5.2-complete.jar'
$ alias grun='java org.antlr.v4.gui.TestRig'
```

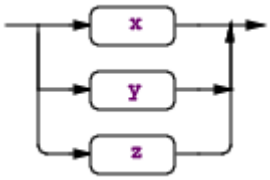
Setup in .bashrc or .bash_profile

Example Grammar File (*.g4)

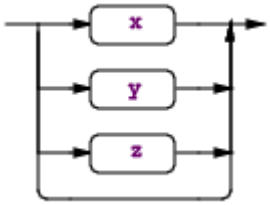
```
/* Example grammar for Expr.g4 */
grammar Expr;               // name of grammar

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

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

Regular expression (subrules)

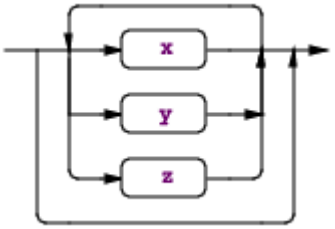
- `(x|y|z)` : match any alternative within the subrule exactly.



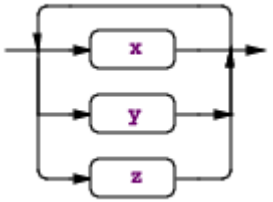
- `(x|y|z)?` : match nothing or any alternative within subrule.



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



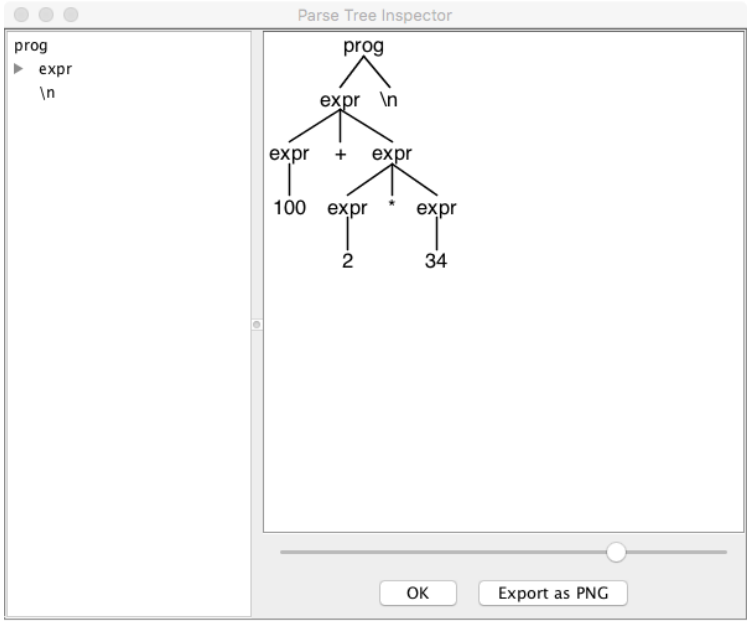
- `(x|y|z)+` : match an alternative within subrule one or more times.



Running ANTLR Parser Generator

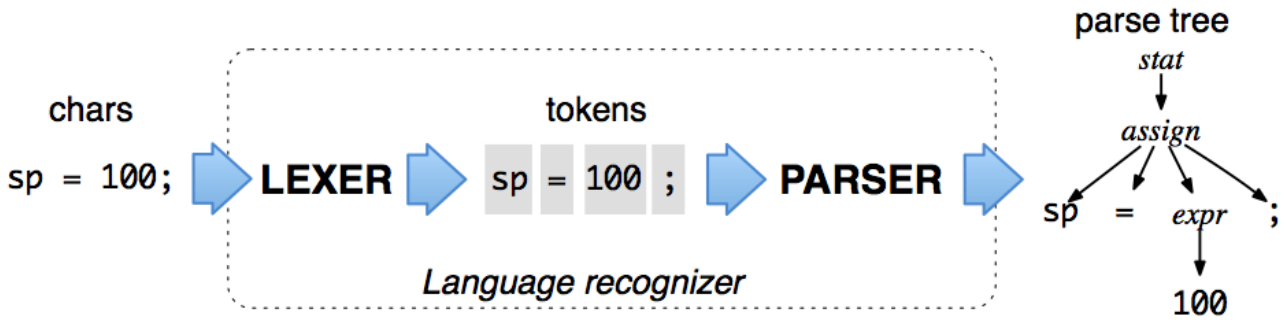
- Writing a grammar file
  - E.g., `Expr.g4` (slide 4)
- Process with ANTLR
  - `$ antlr4 Expr.g4`
- Compile java programs
  - `$ javac Expr*.java`
- Run a generated parse
  - `$ grun Expr prog -gui`
  - `$ grun Expr prog -tree`

```
$ 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



Parse Tree Manipulation

- Now, you have a parse tree.
  - Walk a parse tree with ANTLR tools – Listener or Visitor
  - Listener
    - Walk all parse tree with DFS from the first root node
    - Make functions triggered at entering/exit of nodes
    - *e.g.,* `ExprBaseListener.java` is generated from `antlr4`
  - Visitor
    - Make functions triggered at entering/exit of nodes.
    - Unlike listener, user explicitly call visitor on child nodes
    - To generate visitor class, use `-visitor` option for `antlr4`
      e.g., `$ antlr4 -no-listener -visitor Expr.g4`

ExprBaseListener.java

```
// Generated from Expr.g4 by ANTLR 4.5.2

import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.tree.ErrorNode;
import org.antlr.v4.runtime.tree.TerminalNode;

/**
 * This class provides an empty implementation of {@link ExprListener},
 * which can be extended to create a listener which only needs to handle
 * a subset of the available methods.
 */
public class ExprBaseListener implements ExprListener {
   @Override public void enterProg(ExprParser.ProgContext ctx) { }
   @Override public void exitProg(ExprParser.ProgContext ctx) { }
   @Override public void enterExpr(ExprParser.ExprContext ctx) { }
   @Override public void exitExpr(ExprParser.ExprContext ctx) { }

   @Override public void enterEveryRule(ParserRuleContext ctx) { }
   @Override public void exitEveryRule(ParserRuleContext ctx) { }
   @Override public void visitTerminal(TerminalNode node) { }
   @Override public void visitErrorNode(ErrorNode node) { }
}
```

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

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

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

ExprEvalApp.java

```
class EvalListener extends ExprBaseListener {
   // hash-map for variables' integer value for assignment
   Map<String, Integer> vars = new HashMap<String, Integer>();

   // stack for expression tree evaluation
   Stack<Integer> evalStack = new Stack<Integer>();

   @Override
   public void exitProg(ExprParser.ProgContext ctx) {
      System.out.println("exitProg: ");
   }

   @Override
   public void exitExpr(ExprParser.ExprContext ctx) {
      System.out.println("exitExpr: ");
   }

   @Override
   public void visitTerminal(TerminalNode node) {
      System.out.println("Terminal: " + node.getText());

      //
      //Integer v =Integer(node.getText());
      //evalStack.push(v);
   }

}
```

```
public class ExprEvalApp {
   public static void main(String[] args) throws IOException {
      System.out.println("** Expression Eval w/ antlr-listener **");

      Console c = System.console();
      if (c == null) {
         System.err.println("No Console");
         System.exit(1);
      }
      String input = c.readLine("Input: ");
      input += '\n';

      // Get lexer
      ExprLexer lexer = new ExprLexer(new ANTLRInputStream(input));
      // Get a list of matched tokens
      CommonTokenStream tokens = new CommonTokenStream(lexer);
      // Pass tokens to parser
      ExprParser parser = new ExprParser(tokens);
      // Walk parse-tree and attach our listener
      ParseTreeWalker walker = new ParseTreeWalker();
      EvalListener listener = new EvalListener();

      // walk from the root of parse tree
      walker.walk(listener, parser.prog());
   }
}
```

Programming Assignment #1 (Calculator)

- Build a Java program using ANTLR **Listener** class
  - Expand Expr.g4
    - accept multiple assignments and expressions terminated with ';'
    - calculate the resulting values of expressions
    - Add grammar to accept assignment of value to variables (e.g., a = 100)

```
prog : (assn ';' NEWLINE?| expr ';' NEWLINE?)*
assn : ID '=' INT ;
ID : [a-zA-Z]+ ;
```

PA#1 (cont'd)

- Modify *ExprEvalApp.java* to do the following
  - accept input from *file-path* at command line
  - perform expression-tree evaluation by adopting shunting-yard algorithm with slight modifications
  - print out resulting value

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

// parser rules
prog : (assn ';' NEWLINE? | expr ';' NEWLINE?)*;
expr : expr ('*'|'/') expr
     | expr ('+'|'-') expr
     | INT
     | ID
     | '(' expr ')'
     ;
Assn : ID '=' INT
     ;

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

Grading Policy

- Discussion is allowed, but plagiarism is not allowed
  - If any of the codes is **copied from elsewhere** (e.g. your friends or Google), you'll get absolutely **0 points** (no mercy, no exception).

- This matter applies equally to all the projects afterwards.

Reference

- The Definitive ANTLR 4 Reference - Terence Parr

- http://antlr.org > Dev Tools > Resources
  - Documentation
    - https://github.com/antlr/antlr4/blob/master/doc/index.md
  - Runtime API > Java Runtime (release 4.5.2)
    - http://www.antlr.org/api/Java/Index.html

- Java util package
  - www.tutorialspoint.com/java/util/index.htm