SUNG KYUN KWAN UNIVERSITY

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)

Install ANTLR (version 4.8) – Java tools

- ANTLR (www.antlr.org)
  - https://www.antlr.org/download/antlr-4.8-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-4.8-complete.jar -o
antlr-4.8-complete.jar

$ vi ~/.bashrc
export CLASSPATH=”.:/usr/local/lib/antlr-4.8-complete.jar:$CLASSPATH”
alias antlr4=’java -jar /usr/local/lib/antlr-4.8-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

Download (ANTLR version 4.8) – C++ runtime

- ANTLR download page (https://www.antlr.org/download.html)
  - Under C++ Target, download source for Linux
    - https://www.antlr.org/download/antlr4-cpp-runtime-4.8-source.zip
- Compile from the source and install to /usr/local

```
$ sudo apt install build-essential zip cmake uuid-dev pkg-config
$ unzip antlr4-cpp-runtime-4.8-source.zip –d ANTLR4
$ cd ANTLR4
$ mkdir build && mkdir run && cd build
$ cmake ..
$ DESTDIR=../run make install

$ cd ../run/usr/local/include
$ sudo cp –r antlr4-runtime /usr/local/include
$ cd ../lib
$ sudo cp * /usr/local/lib
$ sudo ldconfig

$ vi ~/.bashrc
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/
$ source ~/.bahsrc
```

Add this line at the end of ~/.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

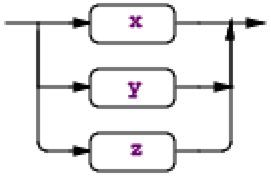
- . matches any single character
- * matches zero or more copies of preceding expression
- + matches one or more copies of preceding expression
- ? matches zero or one copy of preceding expression
  - -?[0-9]+ : signed numbers including optional minus sign
- [ ] matches any character within the brackets
  - [Abc1], [A-Z], [A-Za-z], [^123A-Z]

exclude [123A-Z]

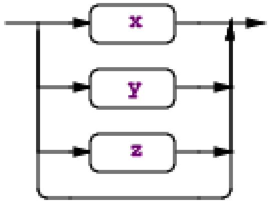
- ^ matches the beginning of line
- $ matches the end of line
- \ escape metacharacter e.g. \* matches with *
- | matches either the preceding expression or the following
  - abc|ABC
- ( ) groups a series of regular expression
  - (123)(123)*

Regular expression (subrules)

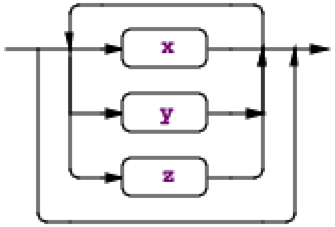
- (x|y|z) : match any alternative within the subrule exactly



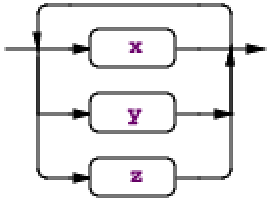
- (x|y|z)? : match nothing or any alternative within subrule



- (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 5)

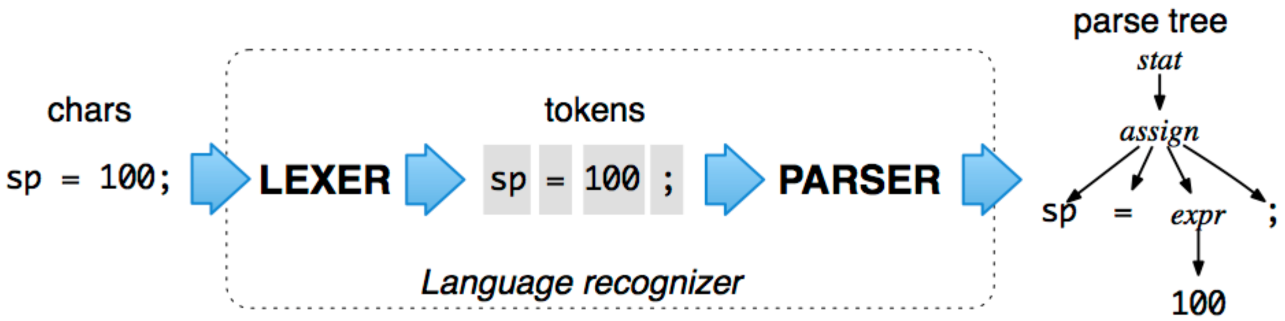
- Process with ANTLR for C++

```
$ antlr4 -Dlanguage=Cpp Expr.g4
$ ls *.cpp *.h
 ExprBaseListener.cpp   ExprBaseListener.h
 ExprListener.cpp       ExprListener.h
 ExprLexer.cpp          ExprLexer.h
 ExprParser.cpp         ExprParser.h
```

- Now we are ready to write main program

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.cpp/h` 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`

```
$ antlr4 -Dlanguage=Cpp -no-listener -visitor Expr.g4
```

ExprBaseListener.cpp/h

```
// Generated from Expr.g4 by ANTLR 4.8
#pragma once
#include “antlr4-runtime.h”
#include “ExprListener.h”

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

   virtual void enterEveryRule(antlr4::ParserRuleContext * /*ctx*/) override { }
   virtual void exitEveryRule(antlr4::ParserRuleContext * /*ctx*/) override { }
   virtual void visitTerminal(antlr4::tree::TerminalNode * /*node*/) override { }
   virtual void visitErrorNode(antlr4::tree::ErrorNode * /*node*/) override { }
};
```

```
/* 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;
```

ExprBaseListener.cpp/h: generated by ANTLR4 along with multiple cpp/h files and others

ExprMain.cpp (user code)

```
#include <iostream>
#include “ExprBaseListener.h”
#include “ExprLexer.h”
#include “ExprParser.h”

using namespace std;
using namespace antlr4;
using namespace antlr4::tree;

class EvalListener : public ExprBaseListener {
   // C++ STL map for variables' integer value for assignment
   map<string, int> vars;
   // C++ STL stack for expression tree evaluation
   stack<int> evalStack;

   // add more fields you need …

public:
   virtual exitProg(ExprParser::ProgContext *ctx) {
      cout << "exitProg: " << endl;
   }

   virtual exitExpr(ExprParser::ExprContext *ctx) {
      cout << "exitExpr: " << endl;
   }

   virtual visitTerminal(tree::TerminalNode node) {
      cout << "Terminal: " << node->getText() << endl;

      //if (node->getSymbol()->getType() == ExprLexer::INT) {
      //   int v = atoi(node->getText().c_str());
      //   evalStack.push(v);
      //}
   }
   // add more methods you need …
};
```

example class for expression evaluation

skeleton code for listener based application

```
int main() {
    cout << “** Expression Eval with ANTLR listener **”);

    ANTLRInputStream input(cin);       // set up input
    ExprLexer lexer(&input);           // Get lexer
    CommonTokenStream tokens(&lexer); // Get a list of tokens
    ExprParser parser(&tokens);        // Pass tokens to parser
    ParseTree *tree = parser.prog();  // Get parse tree

    // Print tree in Lisp style
    cout << tree->toStringTree(&parser) << endl;

    // Walk parse-tree and attach our listener
    ParseTreeWalker walker;
    EvalListener listener;

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

Programming Assignment #1 (Calculator)

- Build a C++ 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 assignments of values to variables (e.g., a = 100)

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

PA#1 (cont’d)

- Modify *ExprMain.cpp* & *Expr.g4* to do the following
  - accept input from *file-path* at command line
  - perform expression-tree evaluation by using shunting-yard algorithm and converting to postfix notation
  - Should use ONLY listener, NOT visitor
  - For assn, use std::map class
  - print out resulting value
    - calculation should be in *double*
    - 5 / 2 => 2.5 not 2
- Run your app with input.txt

```
$ cat input.txt
a = -100;
a + -2.1 * 34;
10 * (+5.0/2);
$ ./expr.exe input.txt
-171.4
25.0
$
```

```
/* Expr.g4 extended - Modify this for PA#1 */
grammar Expr;

// parser rules
prog : (assn ’;’ NEWLINE? | expr ’;’ NEWLINE?)*;
expr : expr (’*’|’/’) expr
     | expr (’+’|’-’) expr
     | num
     | ID
     | ’(’ expr ’)’
     ;
assn : ID ’=’ num
     ;
num  : INT
     | REAL
     ;

// lexer rules
NEWLINE: [\r\n]+ ;
INT: [0-9]+ ;              // should handle negatives
REAL: [0-9]+’.’[0-9]* ; // should handle signs(+/-)
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 internet),
    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 (look into “Java Runtime” for ANTLR4 APIs)
    - http://www.antlr.org/api/

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

- Other resource (C++ target)
  - https://tomassetti.me/getting-started-antlr-cpp/
  - http://www.cs.sjsu.edu/~mak/tutorials/InstallANTLR4Cpp.pdf
- C++ STL tutorials
  - https://www.studytonight.com/cpp/stl/
  - https://www.cppreference.com/Cpp_STL_ReferenceManual.pdf