

CMPE 152: Compiler Design

October 24 Class Meeting

Department of Computer Engineering
San Jose State University



Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak



ANTLR 4 Review

- ❑ Feed ANTLR a .g4 grammar file.
- ❑ ANTLR generates (in Java or C++):
 - a parser
 - a lexer (scanner)
 - parse tree utilities
- ❑ Therefore, for your compiler projects, you don't have to write that code.
- ❑ You must have a correct grammar file.

Example ANTLR Grammar File

Expr.g6

```
grammar Expr;

/** The start rule; begin parsing here. */
prog:  stat+ ;

stat:  expr NEWLINE
      | ID '=' expr NEWLINE
      | NEWLINE
      ;

expr:  expr ('*' | '/') expr
      | expr ('+' | '-') expr
      | INT
      | ID
      | '(' expr ')'
      ;
```

t.expr

```
193
a = 5
b = 6
a+b*2
(1+2)*3
```

Tokens

```
ID  :  [a-zA-Z]+ ;          // match identifiers <label id="code.tour.expr.3"/>
INT  :  [0-9]+ ;           // match integers
NEWLINE: '\r'? '\n' ;      // return newlines to parser (is end-statement signal)
WS   :  [ \t]+ -> skip ;   // toss out whitespace
```

Java Command Line

```
Expr-Java$ ls
Expr.g4 t.expr
Expr-Java$ antlr4 Expr.g4
Expr-Java$ ls
Expr.g4 ExprBaseListener.java
ExprLexer.tokens ExprParser.java
Expr.tokens ExprLexer.java ExprListener.java t.expr
Expr-Java$ javac Expr*.java
Expr-Java$ grun Expr prog -gui t.expr
```

- **grun** runs a test harness for the grammar.

A Java Main Program for ANTLR

```
public class ExprJoyRide
{
    public static void main(String[] args) throws Exception
    {
        String inputFile = null;
        if (args.length > 0) inputFile = args[0];

        InputStream is = System.in;
        if (inputFile != null) is = new FileInputStream(inputFile);

        ANTLRInputStream input = new ANTLRInputStream(is);
        ExprLexer lexer = new ExprLexer(input);

        CommonTokenStream tokens = new CommonTokenStream(lexer);

        System.out.println("Tokens:");
        tokens.fill();
        for (Token token : tokens.getTokens())
        {
            System.out.println(token.toString());
        }

        ExprParser parser = new ExprParser(tokens);
        ParseTree tree = parser.prog();

        System.out.println("\nParse tree (Lisp format):");
        System.out.println(tree.toStringTree(parser));
    }
}
```

ExprJoyRide.java

Default: ANTLR loads the entire input before processing.

Print the list of tokens.

Print the parse tree in Lisp format.

C++ Command Line

```
Expr-Cpp$ ls
Expr.g4 t.expr
Expr-Cpp$ antlr4 -Dlanguage="Cpp" Expr.g4
Expr-Cpp$ ls
Expr.g4 ExprBaseListener.h ExprLexer.tokens ExprParser.cpp
Expr.tokens ExprLexer.cpp ExprListener.cpp ExprParser.h
ExprBaseListener.cpp ExprLexer.h ExprListener.h t.expr
```

- ❑ Unfortunately, **grun** only works with Java code.
- ❑ If you are a C++ programmer but want to use **grun** to test your grammar file, then first generate the Java code.

A C++ Main Program for ANTLR

```
int main(int argc, const char *args[])
{
    ifstream ins;
    ins.open(args[1]);

    ANTLRInputStream input(ins);
    ExprLexer lexer(&input);
    CommonTokenStream tokens(&lexer);

    cout << "Tokens:" << endl;
    tokens.fill();
    for (Token *token : tokens.getTokens())
    {
        std::cout << token->toString() << std::endl;
    }

    ExprParser parser(&tokens);
    tree::ParseTree *tree = parser.prog();

    cout << endl << "Parse tree (Lisp format):" << endl;
    std::cout << tree->toStringTree(&parser) << endl;

    return 0;
}
```

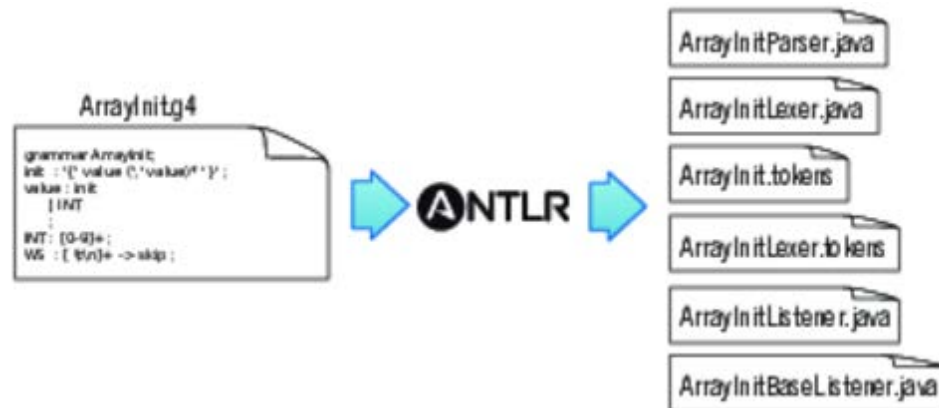
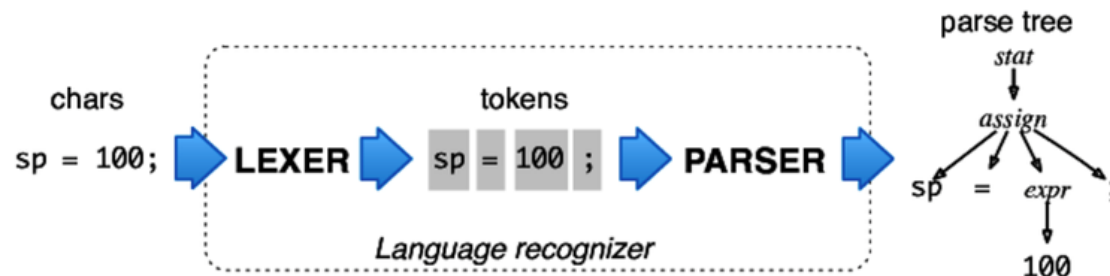
ExprMain.cpp

Default: ANTLR loads the entire input before processing.

Print the list of tokens.

Print the parse tree in Lisp format.

ANTLR Workflow



Syntax Error Handling

- An ANTLR-generated parser has basic syntax error handling and recovery.
 - You can improve the error handling.

```
193
a = 5
b = 6
(a+b*2
(1+2)*3
```

Parse tree (Lisp format):

```
(prog
  (stat (expr 193) \n)
  (stat a = (expr 5) \n)
  (stat b = (expr 6) \n)
  (stat (expr ( (expr (expr a) + (expr (expr b) * (expr 2))) <missing '>'>) \n)
  (stat (expr (expr ( (expr (expr 1) + (expr 2)) )) * (expr 3)) \n))
line 4:6 missing '>' at '\n'
```

Resolving Ambiguities

- Is it a function call as a standalone statement, or a function call in an expression?

```
stat: expr ';'
     | ID '(' ')' ';'
     ;

expr: ID '(' ')'
     | INT
     ;
```

f(); as expression



f(); as function call



Resolving Ambiguities, *cont'd*

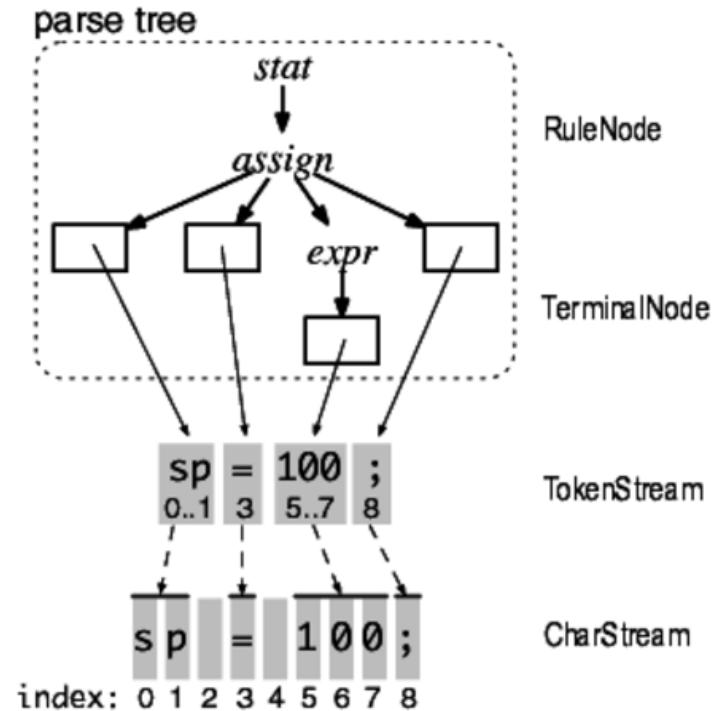
- Is **begin** a reserved word or an identifier?

```
BEGIN : 'begin' ;  
ID    : [a-z]+  ;
```

- ANTLR resolves an ambiguity by choosing the first alternative in the grammar.

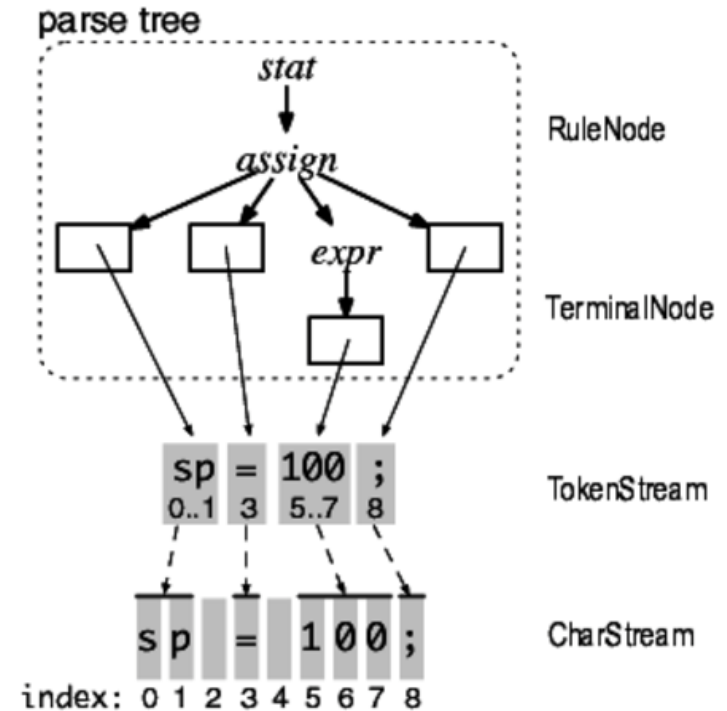
ANTLR Parse Trees

- A token stream is the “pipe” between the lexer and the parser.
- Each token object records the start and stop character indexes into the character stream.



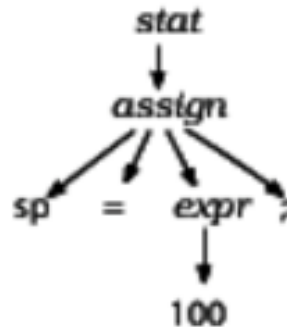
ANTLR Parse Trees, *cont'd*

- ANTLR generates a **RuleNode** subclass for each grammar rule.
- They are called **context objects** because they record everything about the recognition phase of a rule.

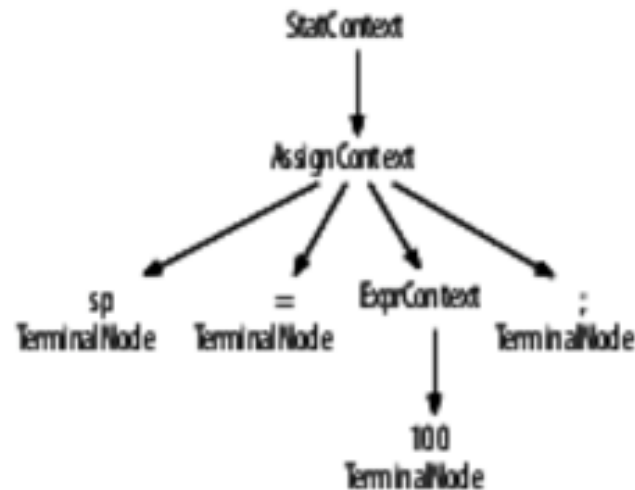


ANTLR Parse Trees, *cont'd*

- The ANTLR-generated parser has corresponding parse tree node class names.



Parse tree



Parse tree node class names

Parse Tree Processing

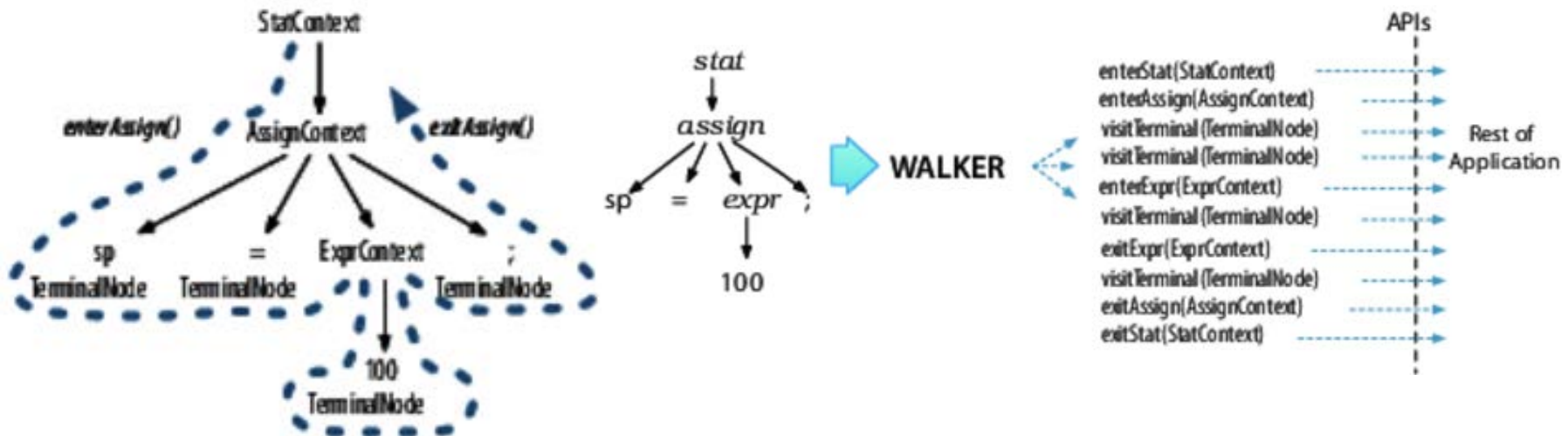
- Recall that after the frontend parser builds the parse tree, it's the backend that processes it.
- ANTLR provides utilities to process the parse tree.
- ANTLR can generate code to process a parse tree with two types of **tree walkers**:
 - listener interface (default)
 - visitor interface

Parse Tree Listener Interface

- ❑ ANTLR generates code that automatically performs a depth-first walk of the parse tree.
 - ❑ You do not have to write a tree walker.
- ❑ It generates a **ParseTreeListener** subclass that is specific to each grammar.
 - The listener class has default enter and exit methods for each rule.
- ❑ You write a subclass that overrides the default enter and exit methods to do what you want.
 - You do not have to explicitly visit child nodes.

Parse Tree Listener Interface, *cont'd*

□ Tree walk and call sequence:

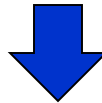


Listener Interface Example

- Parse a Java class definition and generate a Java interface with all the method signatures:

```
public class Demo {  
    void f(int x, String y) { }  
    int[ ] g(/*no args*/) { return null; }  
    List<Map<String, Integer>>[] h() { return null; }  
}
```

demo.java



```
interface IDemo {  
    void f(int x, String y);  
    int[ ] g(/*no args*/);  
    List<Map<String, Integer>>[] h();  
}
```

Listener Interface Example, *cont'd*

- We will use the Java grammar file **Java.g4**.
- Two pertinent methods
 - **classDeclaration**
 - **methodDeclaration**

Java.g4

classDeclaration

```
: 'class' Identifier typeParameters? ('extends' type)?  
  ('implements' typeList)?  
  classBody  
;
```

methodDeclaration

```
: type Identifier formalParameters ('[' ']* methodDeclarationRest  
| 'void' Identifier formalParameters methodDeclarationRest  
;
```

Listener Interface Example, *cont'd*

- The generated **JavaBaseListener.java** has default do-nothing enter and exit methods.

```
public class JavaBaseListener implements JavaListener
{
    ....
    @Override public void enterClassDeclaration(JavaParser.ClassDeclarationContext ctx) {}
    @Override public void exitClassDeclaration(JavaParser.ClassDeclarationContext ctx) {}
    ....
    @Override public void exitMethodDeclaration(JavaParser.MethodDeclarationContext ctx) {}
    ...
}
```

JavaBaseListener.java

Default: Do nothing!

Listener Interface Example, *cont'd*

- ❑ Override the pertinent methods.

ExtractInterfaceListener.java

```
public class ExtractInterfaceListener extends JavaBaseListener
{
    private JavaParser parser;

    public ExtractInterfaceListener(JavaParser parser)
    {
        this.parser = parser;
    }

    @Override
    public void enterClassDeclaration(JavaParser.ClassDeclarationContext ctx)
    {
        System.out.println("interface I" + ctx.Identifier() + " {");
    }

    @Override
    public void exitClassDeclaration(JavaParser.ClassDeclarationContext ctx)
    {
        System.out.println("}");
    }

    ...
}
```

Listener Interface Example, *cont'd*

□ Override the pertinent methods, *cont'd*

ExtractInterfaceListener.java

```
@Override
public void enterMethodDeclaration(
    JavaParser.MethodDeclarationContext ctx)
{
    // need parser to get tokens
    TokenStream tokens = parser.getTokenStream();

    String type = (ctx.type() != null)
        ? tokens.getText(ctx.type())
        : "void";

    String args = tokens.getText(ctx.formalParameters());
    System.out.println("\t" + type + " " +
        ctx.Identifier() + args + ";");
}
```

Listener Interface Example, *cont'd*

ExtractInterfaceTool.java

```
public class ExtractInterfaceTool
{
    public static void main(String[] args) throws Exception
    {
        String inputFile = null;
        if ( args.length>0 ) inputFile = args[0];

        InputStream is = System.in;
        if ( inputFile!=null ) {
            is = new FileInputStream(inputFile);
        }

        ANTLRInputStream input = new ANTLRInputStream(is);
        JavaLexer lexer = new JavaLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        JavaParser parser = new JavaParser(tokens);
        ParseTree tree = parser.compilationUnit();

        ParseTreeWalker walker = new ParseTreeWalker();
        ExtractInterfaceListener extractor = new ExtractInterfaceListener(parser);
        walker.walk(extractor, tree);
    }
}
```

Default parse tree walker.

Listener Interface Example, *cont'd*

□ Equivalent C++ code:

```
class JavaBaseListener : public JavaListener
```

```
{
```

```
public:
```

```
...
```

```
virtual void enterClassDeclaration(JavaParser::ClassDeclarationContext *) override {}
```

```
virtual void exitClassDeclaration(JavaParser::ClassDeclarationContext *) override {}
```

```
...
```

```
virtual void enterMethodDeclaration(JavaParser::MethodDeclarationContext *) override {}
```

```
...
```

```
}
```

JavaBaseListener.h

Default: Do nothing!

Listener Interface Example, *cont'd*

ExtractInterfaceListener.h

```
#include "JavaBaseListener.h"
#include "JavaParser.h"

class ExtractInterfaceListener : public JavaBaseListener
{
public:
    ExtractInterfaceListener(JavaParser *parser);
    virtual ~ExtractInterfaceListener();

    void enterClassDeclaration(JavaParser::ClassDeclarationContext *ctx) override;
    void exitClassDeclaration(JavaParser::ClassDeclarationContext *ctx) override;
    void enterMethodDeclaration(JavaParser::MethodDeclarationContext *ctx) override;

private:
    JavaParser *parser;
};
```

Listener Interface Example, *cont'd*

```
#include <iostream>
#include "ExtractInterfaceListener.h"
#include "antlr4-runtime.h"
```

ExtractInterfaceListener.cpp

```
using namespace std;
using namespace antlr3cpp;
using namespace antlr4;
```

```
ExtractInterfaceListener::ExtractInterfaceListener(JavaParser *parser)
    : parser(parser) {}
```

```
ExtractInterfaceListener::~~ExtractInterfaceListener() {}
```

```
void ExtractInterfaceListener::enterClassDeclaration(
    JavaParser::ClassDeclarationContext *ctx)
{
    cout << "interface I" << ctx->Identifier()->getText() << " {" << endl;
}
```

```
void ExtractInterfaceListener::exitClassDeclaration(
    JavaParser::ClassDeclarationContext *ctx)
{
    cout << "}" << endl;
}
```

Listener Interface Example, *cont'd*

ExtractInterfaceListener.cpp

```
void ExtractInterfaceListener::enterMethodDeclaration(  
    JavaParser::MethodDeclarationContext *ctx)  
{  
    TokenStream *tokens = parser->getTokenStream();  
    string type = (ctx->type() != nullptr)  
        ? tokens->getText(ctx->type())  
        : "void";  
    string args = tokens->getText(ctx->formalParameters());  
    cout << "\t" << type << " " << ctx->Identifier()->getText() << args  
        << ":" << endl;  
}
```

Listener Interface Example, *cont'd*

ExtractInterfaceTool.cpp

```
public class ExtractInterfaceTool
{
    public static void main(String[] args) throws Exception
    {
        String inputFile = null;
        if ( args.length>0 ) inputFile = args[0];

        InputStream is = System.in;
        if ( inputFile!=null ) {
            is = new FileInputStream(inputFile);
        }

        ANTLRInputStream input = new ANTLRInputStream(is);
        JavaLexer lexer = new JavaLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        JavaParser parser = new JavaParser(tokens);
        ParseTree tree = parser.compilationUnit();

        ParseTreeWalker walker = new ParseTreeWalker();
        ExtractInterfaceListener extractor = new ExtractInterfaceListener(parser);
        walker.walk(extractor, tree);
    }
}
```

Default parse tree walker.

Demo

Assignment #5

- ❑ Write the first draft of the ANTLR 4 grammar file for your source language.
- ❑ Generate a syntax diagram.*
- ❑ Generate the parser and lexer.
- ❑ Compile a sample source program.
- ❑ Generate a parse tree diagram.*
- ❑ *C++ programmers: You will first need to generate Java code in order to use these tools.
- ❑ Due: Tuesday, October 31.