Assignment 0

Due Sep 11, 11:59:59 PM

Introduction

The purpose of this assignment is to get you acquainted with Parsing. In particular, it will help you get familiar with the ANTLR parser generator (www.antlr.org) and the data structures, which you will need to understand to implement parsers.

For this class, we require that you use Unix-like platforms (e.g., Linux or Mac) for development. If Unix is not your primary operating system, please use a virtual machine (e.g., VirtualBox (https://www.virtualbox.org/)). We recommend Ubuntu 14.04.1 (http://www.ubuntu.com/download/desktop) as a relatively pain-free Linux distribution.

Part 0: Getting Started

In this section, we'll get started by downloading ANTLR tool and setting up your development environment. Then we'll test out the installation by running a simple example.

Installing ANTLR

First, create a new directory for this project:

```
$ mkdir csce689-a0
$ cd csce689-a0
```

Download antlr-4.5-complete.jar (http://www.antlr.org/download/antlr-4.5-complete.jar) and add it to classpath:

```
$ sudo curl -O http://www.antlr.org/download/antlr-4.5-complete.jar
$ mv antlr-4.5-complete.jar /usr/local/lib/
$ export CLASSPATH="/usr/local/lib/antlr-4.5-complete.jar:$CLASSPATH"
```

Create aliases for the ANTLR Tool:

```
$ alias antlr4='java -jar /usr/local/lib/antlr-4.5-complete.jar'
$ alias grun='java org.antlr.v4.runtime.misc.TestRig'
```

Now we're ready to use `antlr4` to generate parser and `grun` to test it.

A Simple Example

In directory csce689-a0, put the following grammar inside file `Hello.g4` :

```
// Define a grammar called Hello
grammar Hello;
r  : 'hello' ID ;         // match keyword hello followed by an identifier
ID : [a-z]+ ;             // match lower-case identifiers
WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
```

Then run the ANTLR tool on it:

```
$ antlr4 Hello.g4
$ javac Hello*.java
```

Now test it:

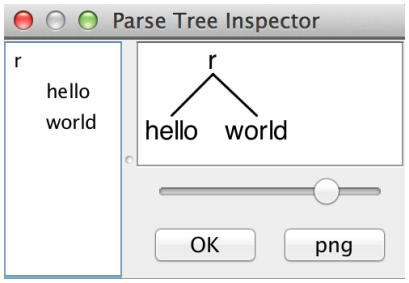
```
$ grun Hello r -tree
hello world
^D
(r hello world)
```

The `^D` means EOF on unix. The `-tree` option prints the parse tree in LISP notation.

You can also look at parse trees visually:

```
$ grun Hello r -gui
hello world
^D
```

That pops up a dialog box showing that rule `r` matched keyword `hello` followed by identifier `world` .



Part 1: Playing with Java Grammar and Parser

In this section, we'll get a sense of what a real language's grammar and parser look like. When you're done with this part, you should have a basic understanding of Java grammar and parser.

Download the grammar of Java 7 defined in ANTLR language:

```
sudo curl -O https://raw.githubusercontent.com/antlr/grammars-v4/master/java/Java.g4
```

The grammar of Java 7 is something like below:

```
grammar Java;

// starting point for parsing a java file
compilationUnit
    :   packageDeclaration? importDeclaration* typeDeclaration* EOF
    ;

packageDeclaration
    :   annotation* 'package' qualifiedName ';'
    ;

importDeclaration
    :   'import' 'static'? qualifiedName ('.' '*')? ';'
    ;
```

After briefly reading through the grammar, you are ready to generate a parser for it:

```
$ antlr4 Java.g4
```

The generated parser contains the following files:

```
JavaLexer.java          JavaParser.java
Java.tokens             JavaLexer.tokens
JavaBaseListener.java   JavaListener.java
```

Compile the parser code and test it:

```
$ javac *.java
$ grun Java compilationUnit *.java
```

Read the parser code and learn how to use it before go to the next part.

Part 2: Finding Interesting Program Facts through Parsing

We're ready to start writing code. In this section, we'll write a function that finds some interesting facts of a given program using the generated parser. When you're done with this part, you should have a basic sense of static analysis.

You are given a full Java program Test.java (Test.java) that contains a variety of statements. What we are interested in are a special kind of statements: those boolean variables that are used as the condition of `if` statements. For example, in the following code snippets, `quiet diag bail SLL` are such boolean variables.

```
259:    if ( !quiet ) System.err.println(f);

271:    if ( diag ) parser.addErrorListener(new DiagnosticErrorListener());
272:    if ( bail ) parser.setErrorHandler(new BailErrorStrategy());
273:    if ( SLL ) parser.getInterpreter().setPredictionMode(PredictionMode.SLL);
```

Your task is to write code to find out such boolean variables with `length > 3` and are only used without `!`. In your code you need to print out their name and line number. For example, the output for the code snippets above should be similar to this:

```
diag 271
bail 272
```

You are provided with a sample code JavaParserTest.java (JavaParserTest.java) which has implemented the basic functionality of using the generated parser. For example:

```
CharStream input = new ANTLRFileStream(inputFile); // inputFile is "Test.java"

JavaLexer lexer = new JavaLexer(input);
CommonTokenStream tokens = new CommonTokenStream(lexer);
JavaParser parser = new JavaParser(tokens); //create parser

ParseTree tree = parser.compilationUnit();
ParseTreeWalker walker = new ParseTreeWalker(); // create standard walker
JavaListener listener = new JavaParserTest(); // create a parse tree listener

walker.walk(listener, tree); // traverse parse tree with listener
```

`JavaParserTest` is declared as a subclass of `JavaListener` and overrides the method `enterStatement`:

```
@Override
public void enterStatement(Java7Parser.StatementContext ctx)
{
    System.out.println("enterStatement");
    //your code starts here
}
```

The method above will be called when a Java statement is visited during traversing the parse tree. Your entire code will be written in this method.

Hints

- Google is your friend.
- Check the Java grammar (https://raw.githubusercontent.com/antlr/grammars-v4/master/java/Java.g4) that `if` statement is defined at line 407: `'if' parExpression statement ('else' statement)?` . Hence, the boolean variables we are interested in must be in `parExpression` .
- To determine if the first token in the current StatementContext `ctx` is `'if'` , use the following code:

```
if(ctx.getStart().getText().equals("if")) {
    //first token is 'if'
}
```

Part 3: Rewrite Program Source Code with Parser

Now, let's do one more thing with the parser: add new code to the input program. Specifically, we want to insert printing statements into `Test.java` to print out the name and line number of the boolean variables found in the last part. For example, the following code

```
271:    if ( diag ) parser.addErrorListener(new DiagnosticErrorListener());
```

will be transformed to

```
271:    if ( diag ){
            System.out.println("diag 271");
            parser.addErrorListener(new DiagnosticErrorListener());
        }
```

Hints

- You may use the class `TokenStreamRewriter` to add and print code. Example usage below:

```
TokenStreamRewriter rewriter = new TokenStreamRewriter(tokens);
rewriter.insertBefore(token, "System.out.println("+name+" "+line+");");
System.out.println(rewriter.getText());
```

- The transformed code must be valid. Don't forget to add `"{"` and `"}"` into correct places if needed.
- Don't forget to preserve whitespaces. Google "antlr4 whitespace preserve" to find answers.

Turnin Instructions

1. **You are expected to work through this assigment and commit into a git repo (e.g., your private tamu github repo (https://github.tamu.edu/))**

2. **Share your git repo with `jeffhuang@tamu.edu`.**
3. **The sample code with correct answers will be released after the due time.**