2016编译Project报告

13300200019 吴耀波　13307130316 李旻骏

1. 原理

1.1 词法分析

词法分析阶段是编译过程的第一个阶段。这个阶段的任务是从左到右一个字符一个字符地读入源程序，即对构成源程序的字符流进行扫描然后根据构词规则识别单词(也称单词符号或符号)

1.2 语义分析

语义分析是编译过程的一个逻辑阶段. 语义分析的任务是对结构上正确的源程序进行上下文有关性质的审查, 进行类型审查。

1.3 工具间比较

对比Yacc、Bison和ANTLR三种工具，最大区别在于，yacC/Bison处理LALR文法ANTLR处理LL文法。

Yacc/Bison生成表驱动的解析器，这表示" 字节流" 包含在解析器中的程序的数据则没有这么大的解析器的代码。

ANTLR生成递归下降解析器，这意味着" 字节流"必须包含在解析器的代码中的每个生产规则表示为函数的语法解析器的代码。

2. 代码

2.1 代码结构

- miniJava.g2 miniJava的文法定义文件
- Main.java 程序主入口，利用ANTLR解析目标miniJava文件
- MyListener.java 错误处理监听器，输出错误提示信息
- AstPrinter.java 把ANTLR生成的语法分析树转换为抽象语法树并输出

2.2 核心代码

- **miniJava.g2**

定义了prog, mainclass, classdecl, vardecl, methoddecl, formallist, formalrest, type, statement, exp, explist, exprest, INT, BOOLEAN, ID, WS的基本结构，如：

```
prog:   mainclass classdecl* EOF ;

formallist: type ID formalrest* ;

formalrest: ',' type ID ;

type:   'int' '[' ']'
    |   'boolean'
    |   'int'
    |   ID
    ;

statement:  '{' statement* '}'
         |  'if' '(' exp ')' statement 'else' statement
         |  'while' '(' exp ')' statement
         |  'System.out.println' '(' exp ')' ';'
         |  ID '=' exp ';'
         |  ID '[' exp ']' '=' exp ';'
```

在Terminal中使用ANTLR即可自动生成miniJavaParser等核心类：

```
java -jar /usr/local/lib/antlr-4.5.3-complete.jar -visitor miniJava.g4
javac *.java
```

- **Main.java**

  在Java内使用ANTLR进行解析：

```
    miniJavaLexer lexer = new miniJavaLexer(antlrInputStream);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    miniJavaParser parser = new miniJavaParser(tokens);
    parser.removeErrorListeners();
    parser.addErrorListener(new MyListener());
    new AstPrinter().print(parser.prog().getRuleContext());
```

- **MyListener.java**

  区分词法错误与语义错误并分别进行处理：

```
    @Override
    public void syntaxError(Recognizer<?, ?> recognizer, Object offendingSymbol,
            int line, int charPositionInLine, String msg,
            RecognitionException e) {
        if (msg.startsWith("Lexical Error") || msg.startsWith("Semantic Error")) {
            System.err.println("line " + line + ":" + charPositionInLine + " " +
msg);
            printErrorLine(recognizer, line);
        } else {
            msg = "Syntactic Error:" + msg;
            System.err.println("line " + line + ":" + charPositionInLine + " " +
msg);
            underlineError(recognizer, (Token)offendingSymbol, line,
charPositionInLine);
        }
    }
```

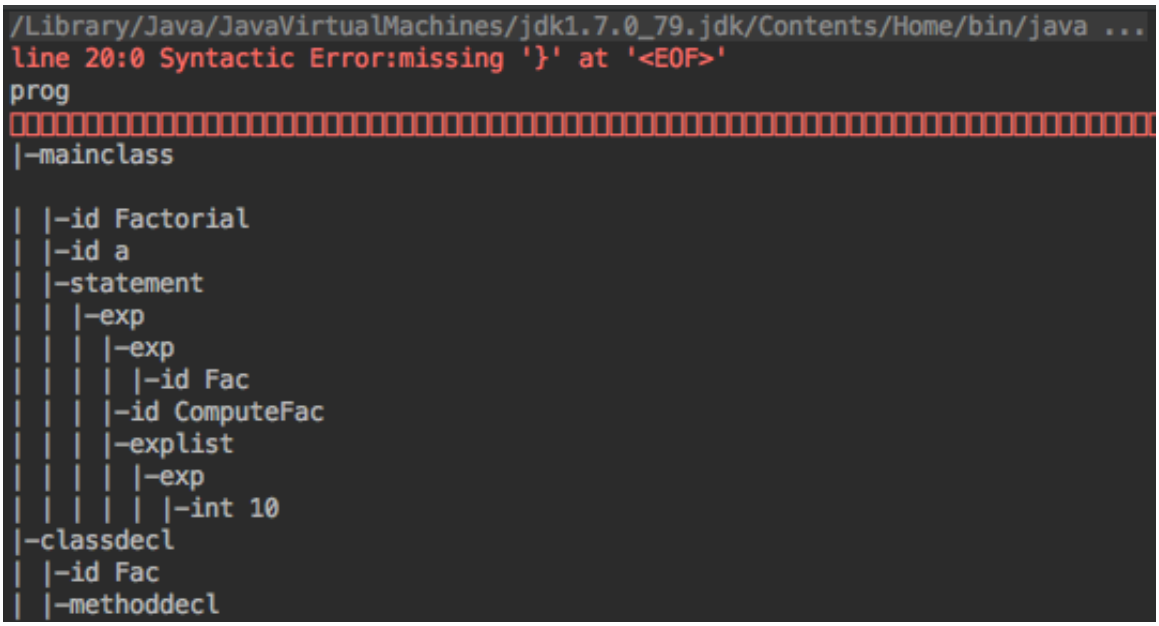
- **AstPrinter.java**

  对ANTLR生成的词法分析树进行遍历，选取所需的结点构造抽象语法树

```
    private void explore(RuleContext ctx, int indentation) {
        String ruleName = miniJavaParser.ruleNames[ctx.getRuleIndex()];
        print_Indentation(indentation);
        System.out.println(ruleName);
        for (int i=0;i<ctx.getChildCount();i++) {
            ParseTree element = ctx.getChild(i);
            if (element instanceof RuleContext) {
                explore((RuleContext)element, indentation + 1);
            } else if (element instanceof TerminalNode) {
                explore_Terminal((TerminalNode) element, indentation + 1);
            }
        }
    }
```

3. 错误处理

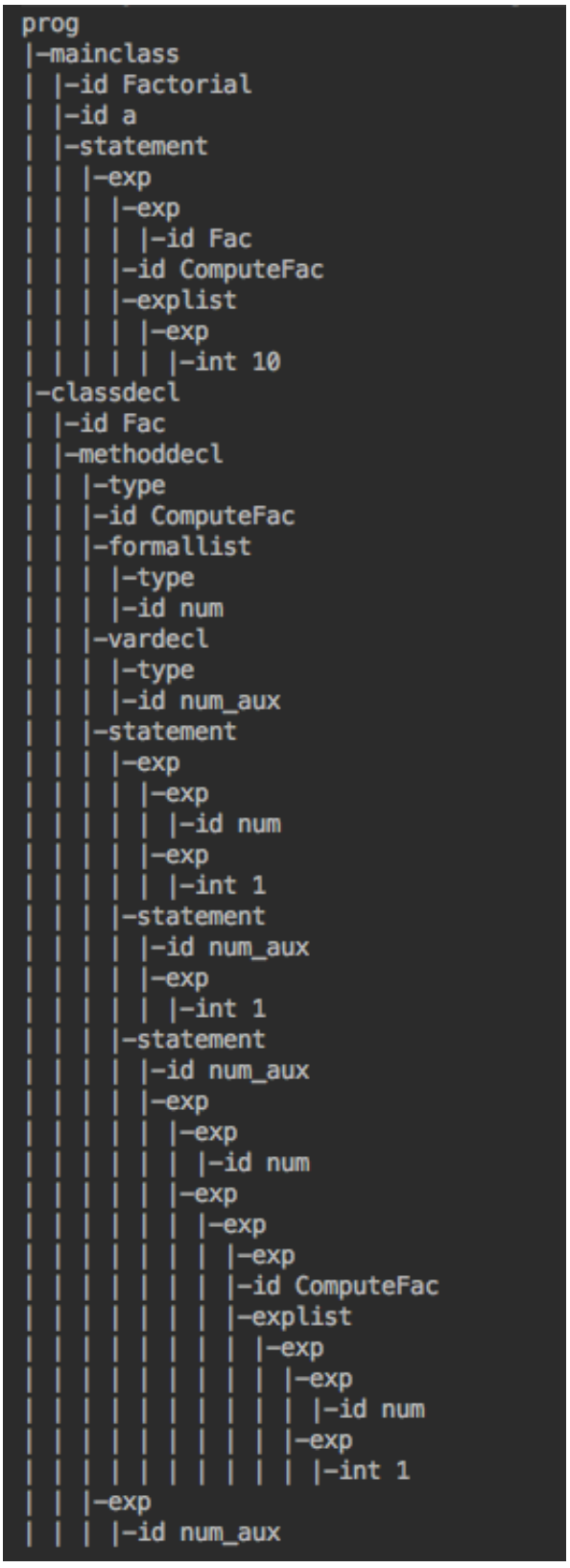
基本按照《The Definitive ANTLR 4 Reference》中第九章的步骤进行错误处理，通过自定义错误监听器MyListener定位到错误处，处理的结果是输出错误所处行以及错误信息。比如缺少'}'时输出提示：



4. 项目感想

坦白说，这个项目我们没有花过多的时间和精力在上面，一分钱一分货我们也知道只是实现了最基本的要求没啥创新，大四了，最后一门专业课，最后一个PJ，还是非常感谢老师和助教！

附：输出截图


prog
|-mainclass
| |-id Factorial
| |-id a
| |-statement
| | |-exp
| | | |-exp
| | | | |-id Fac
| | | |-id ComputeFac
| | | |-explist
| | | | |-exp
| | | | | |-int 10
|-classdecl
| |-id Fac
| |-methoddecl
| | |-type
| | |-id ComputeFac
| | |-formallist
| | | |-type
| | | |-id num
| | |-vardecl
| | | |-type
| | | |-id num_aux
| | |-statement
| | | |-exp
| | | | |-exp
| | | | | |-id num
| | | | |-exp
| | | | | |-int 1
| | | |-statement
| | | | |-id num_aux
| | | | |-exp
| | | | | |-int 1
| | | |-statement
| | | | |-id num_aux
| | | | |-exp
| | | | | |-exp
| | | | | | |-id num
| | | | | |-exp
| | | | | | |-exp
| | | | | | | |-exp
| | | | | | | | |-id ComputeFac
| | | | | | | | |-explist
| | | | | | | | | |-exp
| | | | | | | | | | |-exp
| | | | | | | | | | | |-id num
| | | | | | | | | | |-exp
| | | | | | | | | | | |-int 1
| | |-exp
| | | |-id num_aux