编译 project 报告

李斯哲 13307130370

一. 项目要求

为 MiniJava 语言构造一个编译器前端，将输入的 miniJava 语言代码段转化成抽象语法树。

二. 使用工具选择

在项目中需要使用到词法/语法自动生成工具，常用的工具有 Flex/Bison，Lex/Yacc，Jlex/CUP，JavaCC, ANTLR。下面对这几种不同工具的优缺点做简要分析。

Flex/Bison：Flex 是一个生成词法分析器的工具，它可以利用正则表达式来生成匹配相应字符串的 C 语言代码，其语法格式基本同 Lex 相同。Bison 是属于 GNU 项目的一个语法分析器生成器。Bison 把一个关于 LALR 上下文无关文法的描述转化成可以分析该文法的 C 或 C++ 程序。

这两个工具都是基于 C++的，但 Flex 有一个很严重的缺点，就是 Flex 对源文件的格式要求非常严格，比如若将要求顶行书写的语句变成非顶行书写就会产生致命错误。但他本身对错误的查找能力非常弱，因此如果不是很熟悉的话编写起来会比较困难。

Lex/Yacc：Lex 是 Unix 环境下非常著名的工具,主要功能是生成一个词法分析器的 C 源码，Yacc 是一个经典的生成语法分析器的工具。这两个工具主要是基于 C 语言实现，相较 C++而言更加不方便，同时他们也有着 Flex 工具的缺陷，在源文件编写上比较困难。

JFlex/JavaCUP：JFlex 是 The Fast Lexical Analyser Generator,是 lex/flex 的 java 版本，JFlex 和 JavaCUP 与上述两个工具比较相似，只是是基于 Java 语言的，但由于 Java 语言有更好更方便地工具使用，因此不使用这两个工具。

JavaCC 是一个用 Java 开发的受欢迎的语法分析生成器。这个分析生成器工具可以读取上下文无关且有着特殊意义的语法并把它转换成可以识别且匹配该语法的 Java 程序。相比于前面的工具，他将词法分析和语法分析结合在了一起，

并且其源文件的编写规则更加的简洁。

ANTLR：ANTLR 和 JavaCC 比较相似，但他能够通用于更多的语言，可以为包括 Java，C++，C#在内的语言提供了一个通过语法描述来自动构造自定义语言的识别器，编译器和解释器的框架。相较于 JavaCC 其适用性更广，功能更加强大，因此在这个项目中选择使用 ANTLR 作为词法/语法生成工具。

三. 项目结构及核心代码工作原理

项目主要是由两个部分构成：

第一部分是 MiniJava.g4 文件及 ANTLR 编译后生成的文件，包括 MiniJava.tokens ， MiniJavaBaseListener.java ， MiniJavaLexer.java ， MiniJavaLexer.tokens ， MiniJavaListener.java ， MiniJavaParser.java 。MiniJava.g4 是整个项目核心的代码，是由 MiniJava 语言的 BNF 定义，根据 ANTLR 源文件的编写规则改写而成的，除代码改写外，还需要消除原 BNF 中存在的左递归，主要为 expression 部分

```
Expression  ::= Expression ( "&&" | "<" | "+" | "-" | "*" ) Expression
|   Expression "[" Expression "]"
|   Expression "." "length"
|   Expression "." Identifier "(" ( Expression ( "," Expression )* )? ")"
|   <INTEGER_LITERAL>
|   "true"
|   "false"
|   Identifier
|   "this"
|   "new" "int" "[" Expression "]"
|   "new" Identifier "(" ")"
|   "!" Expression
|   "(" Expression ")"
```

ANTLR 可以将 g4 文件编译为词法分析器，语法分析器和树分析器，并提供了相关接口。

第二部分是 test.txt 测试文件以及 Main.java 函数。

```java
public class Main
{
    public static void main(String[] args) throws Exception
    {
        InputStream is = args.length > 0 ? new FileInputStream(args[0]) : System.in;
        ANTLRInputStream input = new ANTLRInputStream(is);
        MiniJavaLexer lexer = new MiniJavaLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        MiniJavaParser parser = new MiniJavaParser(tokens);
        ParseTree tree = parser.goal();
        JFrame frame = new JFrame("Syntax Tree");
        JPanel panel = new JPanel();
        JScrollPane scrollPane = new JScrollPane(panel);
        TreeViewer viewer = new TreeViewer(Arrays.asList(parser.getRuleNames()),tree);
        panel.add(viewer);
        frame.add(scrollPane);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
        frame.setVisible(true);

    }
}
```

Main 函数通过调用已经编译好的 java 文件的类，并将写有 MiniJava 代码的 test 文件作为输入，可以非常方便的指出 test 文件中的词法错误和语法错误，并能够画出 test 文件中代码的语法树。

四. 错误处理情况

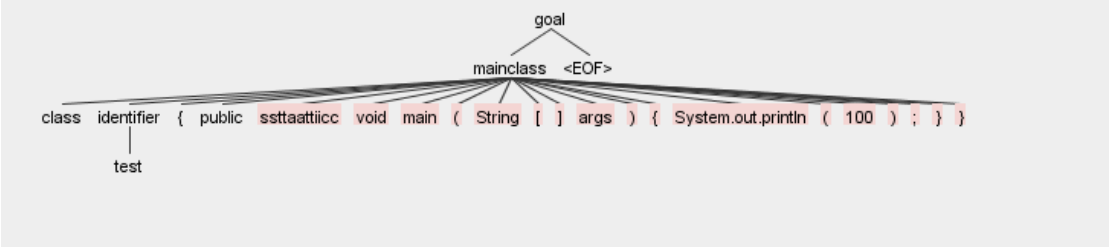
词法错误是指代码中拼写出现错误，一种典型的词法错误是固定单词的拼写错误造成的，比如 test 文件中如下代码

```
class test {
    public ssttaattiicc void main(String[] args)
    {
        System.out.println(100);
    }
}
```

其中 static 明显错误，以 test 文件为输入运行 Main 函数，可以得到错误提示

```
line 2:11 mismatched input 'ssttaattiicc' expecting 'static'
```

，并可以看到绘制成的语法树



其中红色部分为有错误的部分。

语法错误是指代码不符合该语言的语法规范，一种典型的语法错误是大括号或小括号不匹配，或者漏掉分号造成的，例如下图

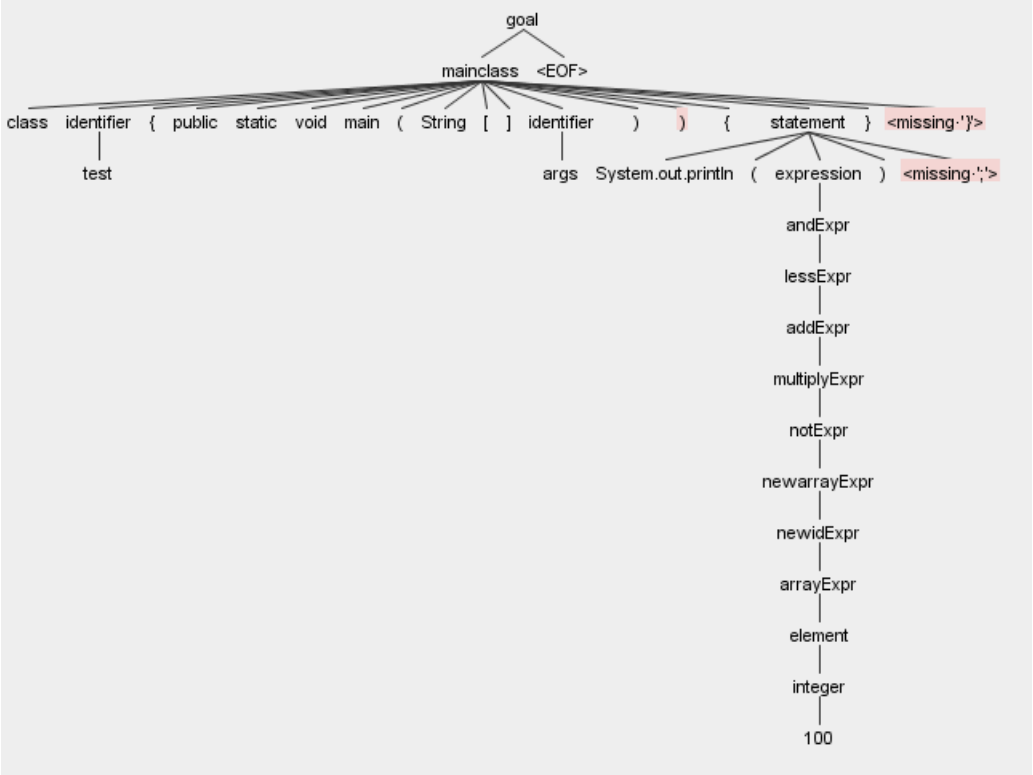
```
class test {
    public static void main(String[] args))
    {
        System.out.println(100)

}
```

第二行中多了一个有括号，第四行漏掉分号，第五行漏电右大括号

错误提示

```
line 2:42 extraneous input ')' expecting '{
line 6:0 missing ';' at '}'
line 10:0 missing '}' at '<EOF>'
```

其语法树



语义错误一般指代码逻辑上的错误，暂时无法进行处理。

五．额外功能与项目感想

因为个人能力原因和时间原因，在项目中没有实现额外的功能，仅仅使用强大的 ANTLR 工具实现了简单的词法，语法错误的处理。

通过这次的项目，让我对课堂中学到的知识有了更清晰的认识。在学习过程中会感觉语法部分非常的抽象，很难能够理解，但是在将 BNF 文件改写为 g4 文件的过程中对语法有了比较形象化的了解，同时由于需要消除 BNF 中的左递归，对消除左递归的方法也进一步的运用，在这次项目也学到了很多课堂中没有的知识。