MiniJava编译报告

小组成员：

罗思成

14307130032

杜昉

14307130157

一、 编译原理简介

本项目实现针对MiniJava 语言的编译器的前端，输出抽象语法树，主要包括了词法分析、语法分析、语义分析。其中词法/语法分析主要由词法/语法自动生成工具ANTLR完成，语义分析包括了变量声明检查、类型检查等。

二、 项目分工

罗思成：

环境配置，Antlr代码的书写与语法树的生成，部分语义错误，错误优化输出。

杜昉：

对语义错误部分的内容进行构架，将type、method等进行再定义，部分语义错误的编写，修正.g4文件中的内容，

三、 实验工具及参考文件

Itellij Idea， ANTLR，Github(Link below)

《The Definitive ANTLR 4 Reference, 2nd Edition》

https://github.com/MrPBDu/Compile-miniJava.git

四、 ANTLR工具和文法的定义

ANTLR主要分了两部分，parser rule和lexer rule。需要注意的是区分这两种rule的方式是大小写，小写开头的rule是parser rule，大写开头的则是lexer rule。匹配规则时，按照从上往下的顺序进行匹配，匹配第一个合法规则。

ANTLR 还有一些需要用到的技巧：比如说对于细分的文法，可以通过加#表示tag，在生成parser时生成器会对该rule的每一条细分文法都进行细分，对于后面的工作有帮助。

这里也提一个在构建时遇到的有趣的case。当时搭环境的时候和隔壁室友同时遇到了左递归的问题，但我们使用的环境不同，我们选择的是IntelliJ IDEA，而他选择的是虚拟机。而我们仅仅只是将.g4文件中的Identifier部分改成大写，左递归就自己消失了，而那位同学却不行，需要自己重新修改一边BNF的内容。

五、 源代码的架构和工作原理

这边因为直接使用的是IDEA，因此文件的输入相对而言不如用console界面而言来得方便，这是值得改进的一点地方。用FileInputSteam读入数据，这边存在一点疑惑的地方，miniJava和testcase这两个上级目录应该是存在同一个路径下，但是我们使用“testcase\factorial.java”这一相对路径的话，无法识别到这一路径。

```
File file = new File( pathname: "D:\\download\\factorial.java");
FileInputStream fis = new FileInputStream(file);
ANTLRInputStream input = new ANTLRInputStream(fis);
```

主程序的代码框架test.java（相当于main函数，IDEA可直接选择），我们从取源程序，先进行词法分析(lexer)得到了对应的tokens，然后将tokens作为输入，得到相应的parser，由这个parser 建立出一棵语法树。

BuildPhase和CheckPhase就是两个继承自MiniJavaBaseListener 的类，用

于遍历语法树。接来下的部分是对这棵语法树进行两次遍历，进行语法语义检查。

为此，定义了2个枚举类型(VarType, Type)，4 个Symbol类(Symbol, VarSymbol, MethodSymbol, ClassSymbol)，一个Range接口和对应的Block类。

Symbol表示符号，其对应的SymbolType 有三种：类、方法、变量，

Symbol具有name和symbolType两个基本属性。

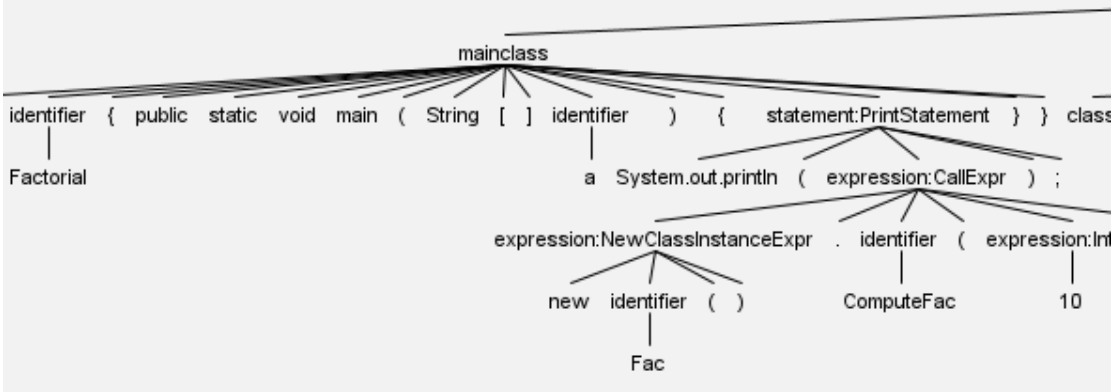
Range 定义了作用域的接口，Block 实现了该接口。Block 内以 HashMap 的形式存放了当前作用域里的符号表 symbols。同时 Block 是一个树节点，有 ParentRange 和 classSymbol 属性，ParentRange 用于存放上一层的域，classSymbol 表示当前类。lookup 方法。该方法检测当前域附带的符号表 symbols 里是否含有该符号。若没有，则向上逐层查找符号表，直到最外层。

ClassSymbol表示了一个类的符号，没有额外的属性。

VarSymbol表示变量的符号，变量有对应的VarType。变量是某个Class的实例时，VarType为typeclass，ClassSymbol记录类。

MethodSymbol表示了方法的符号，方法的返回类型由VarType 存储。同样，方法返回类型可能是一个类，要存储这个类的ClassSymbol。

生成抽象语法树，则在.g4文件中“Test Rule Goal”，则可看到。



上图为“Factorial.java”文件对应的抽象语法树（部分）

六、 错误处理机制

ANTLR能自动检测词法错误和部分语法错误，且能够进行一个token 范围的错误修复，可以继承ErrorListener重写，更多增加了语义错误的检查。

错误处理机制遍历两遍语法树，分别为BuildPhase和CheckPhase，第一遍定义类和方法，第二遍进行错误检查。

这两个类都继承自MiniJavaBaseListener 类。

BuildPhase拥有globalBlock，currentRange，range，classRange四个变量。除了currentRange用于获取代码的作用域，另三个变量均会作为参数传递给CheckPhase的构造器。CheckPhase与他变量结构基本相同但是功能不同。

下面通过一个实际例子，对实现的常用几类语义错误进行详解和演示。

1. 重复定义

在 BuildPhase 中的 enterVardeclaration 中进行，在变量定义前先判断当前域内是否有同名变量。

2. 未定义变量

在 CheckPhase 中的 enterIdentifier 中进行，在变量使用时判断当前域内是否有该变量。

这两者代码如下。

```
@Override
public void enterIdentifier(miniJavaParser.IdentifierContext ctx) {
    try {
        String varname = ctx.IDENTIFIER().getText();

        if (currentRange.lookup(varname) == null) {
            printError(ctx.IDENTIFIER().getSymbol(), msg: "variable or class not defined '" + varname + "'");
        }
    } catch (Exception e) {
        printError(ctx.getStart(),  msg: "Compiler met an unrecognizable error.");
    }
}
```

其中 ctx.IDENTIFIER.()getText()能读入当前所使用的变量的类型名，然后 lookup（）则能够从之前 Build 过程中存储的所有类型名中查找。如果没有找到，报错如下

```
line 10:5 Compiler met an unrecognizable error.
    int int;
     ^
line 11:5 variable or class not defined 'num'
    if (num < 1)
     ^
line 12:5 variable or class not defined 'num_aux'
       num_aux = 1 ;
     ^
```

七、 优化报错系统

继承ErrorListener重写，对于词法，语法错误进行优化，显示具体的行数，位置以及错误信息。有书上的UnderlineListener代码，显示具体错误行。实现词法部分的不可识别字符的显示，对于后续的语义错误也有相同的输出。

```
rule stack: [goal, classdeclaration, methoddeclaration, vardeclaration
line 10:5 mismatched input 'int' expecting IDENTIFIER
    int int;
     ^^^
rule stack: [goal, classdeclaration, methoddeclaration, statement]
line 13:1 missing 'else' at 'elsee'
    elsee
 ^^^^^
```

八、 遇到的问题和解决

1.由于关于Identifier存在性的问题是在enterIdentifier时检测的导致一些问题。

如何调用另一个类的方法？

样例代码中，MainClass会调用后面Class的方法。但该类的方法不在当前的作用域内。因此identify存在性的检测中会出现ComputeFac的错误。可以在第一遍遍历时定义类和方法时，保存了一个<ClassSymbol, Range>的Map，供其他类调用这个类的方法和变量，没有具体实现。

2.继承的变量处理

由于其继承了未定义的类，因而也会被我的编译器误检测到，出现同一个词的继承类找不到和变量名无法找到两个错误，重复报错。

九、 实验感想

这次实验内容还是比较充实，虽然一开始依靠老师提供的BNF内容，可以直接用ANTLR工具生成词法、语法的部分，但是当要写语义的时候才发现，大部分内容要需要靠自己再次定义、排布、调用函数等。也对Java有了一定深入的了解，顺带也复习了一点java的词法、语法和语义。

当然也遇到了许许多多的问题，有些解决了，有些还仍待解决。

另外就是和同学合作上也学习到了许多新的内容。比如，因为这次使用git合作嘛，就会遇到一个问题，两个人同时在写一个文件，结果一个人写完后直接就push上去，导致另一个人得merge的工作（特别是写的内容有相关时，merge是个挺费力的工作）。这也是一些交流、安排上的收获。