# 基于ANTLR的MiniJava前端

项目报告

郑哲东 12307130083
韩韵衡 12307130140

2015.12

一．项目目标：

使用词法/语法自动生成工具ANTLR，为miniJava语言构造一个编译器的前端，将输入的miniJava语言转化成抽象语法树。对原有语法做一点扩充，以及加入对错误（词法错误、语法错误、语义错误等）的提示和修复。

二．项目简介：

使用了BNF语法来构建Minijava前端（在附件中给出的Minijava.g4为语法文件）。

三.实验数据：

关于Minijava的6个测试程序(包含树 / 数组搜索等) 以及
test.java语法树测试程序

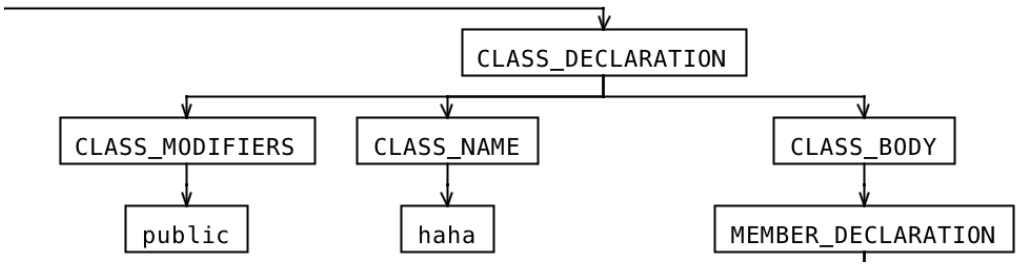
四.实验环境：

antler3.2 + antlrwork1.4 + Java1.7.0_80 (错误检测中使用到了 antlr4)

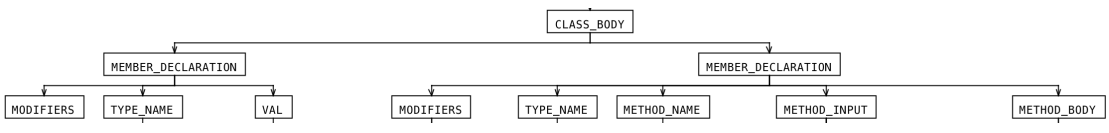
五.实验方法：

**1.基本块的描述：**

-a.class声明：class 由 class_modifier/class_name/class_body组成，通过词法分析，将每一部分分隔开，进行子树分析的操作。




-b.member声明：member为class_body中的函数或变量声明。若为变量声明，则通过varDeclaration分析分为平行的三个子节点var_modifier/var_type/var_assign,若为函数则通过methodDeclaration分析为method_modifier/method_type/method_name/method_input/method_body三个子节点。(下图的例子中为一个类内变量声明 / 一个函数声明)

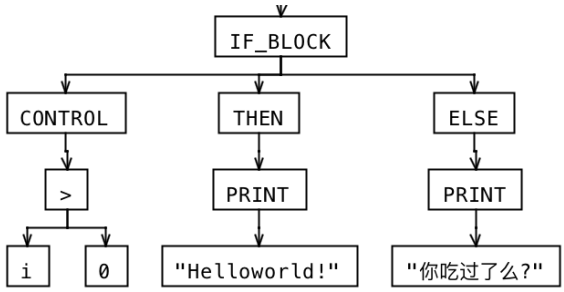


c. method声明：method中可能含有多句statement（如赋值语句／条件语句／循环语句）。将分为多个block子节点，如IF_BLOCK，WHILE_BLOK,FOR_BLOCK等进行进一步分析。下一部分将具体解释。

**2.语句块的描述：**

-a.IF_BLOCK:

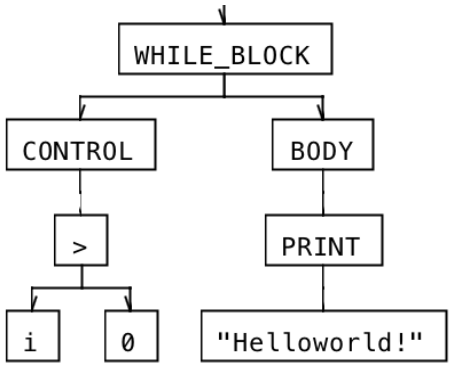
IF语句由条件和执行部分组成，将其分为三个子节点；




(注：此处“”双引号不能省略，表示字符串。比如print it表示打印变量，print “it”为打印字符串，不用引号会有歧义)

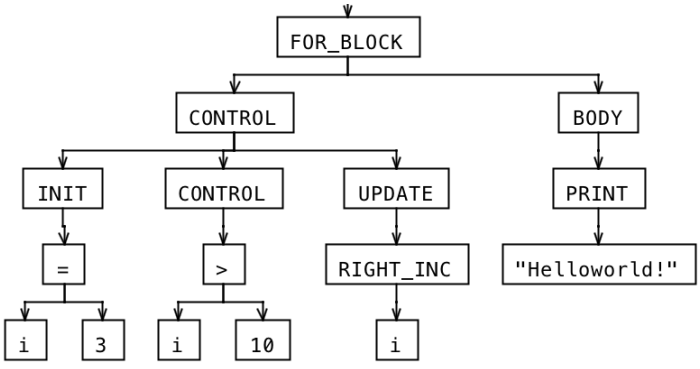
-b.WHILE_BLOCK:

WHILE语句由条件和循环部分组成，将其分为两个子节点；




-c.FOR_BLOCK:

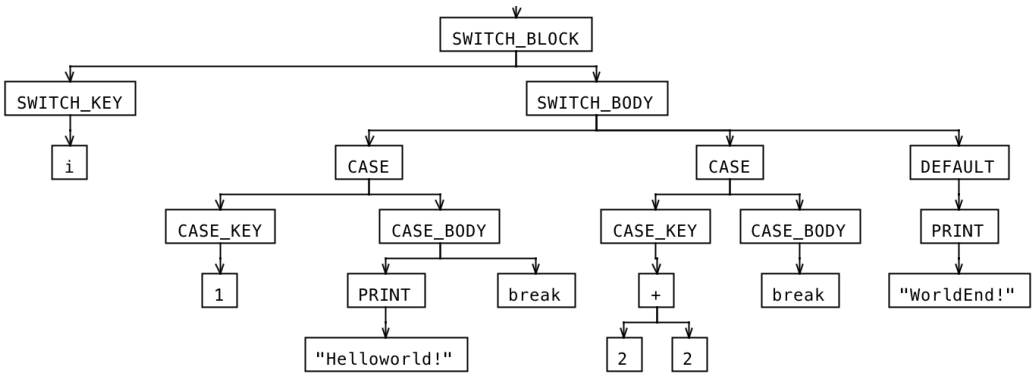
FOR语句由条件和循环部分组成，将其分为两个子节点；条件内部再分为INIT/CONTROL/UPDATE



(注：由于有左++和右++如i++和++i，为避免歧义故设了tokenRIGHT_INC/DEC和LEFT_INC/DEC来区分)

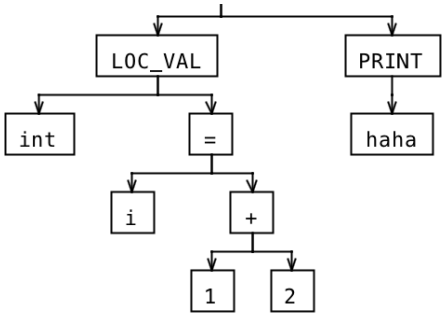
-d.SWITCH_BLOCK:

SWITCH由多个CASE和一个DEFAULT 组成。分为key和body执行部分。




-e.赋值语句,单目多目运算符:

赋值语句将其按等式左右分为两个子节点；单目运算符一个子节点；多目并列多个子节点（下例为局部变量赋值 int i = 1+2 以及可视为单目运算的System.out.print(haha)）




**3.相比java简化的是:**

-a.为了防止初学者弄混，同时也没必要引入太多太全的package / library，故本项目中实现的minijava简化java的library，考虑到在它的standard core中已包含了17个常用class。故没有加入对包检测(import)及(package)；

-b.按照［1］文档的说明，Java需要对每个可能异常的方法实现异常处理，这会增加初学者的学习难度和代码长度。故为了简化，try catch没有加入本项目的语法检测；

-c.按照［1］文档的说明，do-while语句对初学者可能产生buggy的程序，而同样的内容可以通过while来实现，故do-while也没有加入语法检测；

-d.继承extend/接口implement也没有加入。

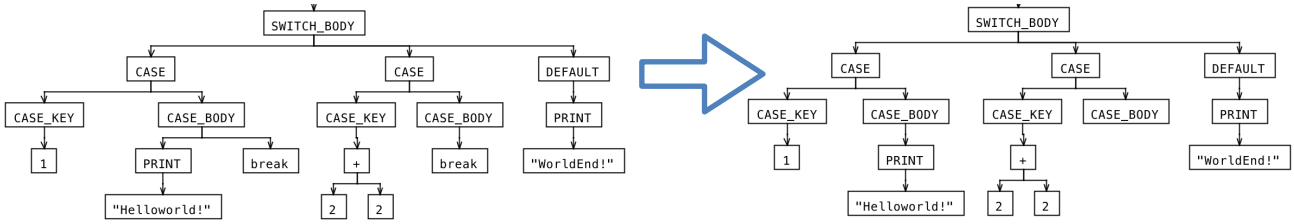
**4.本项目增加的有:**

-a.main class 检测

考虑到每一个java文件中可能没有main method，但只有一个public class或者是程序入口或者是为了供其它java文件调用。故由于public class的特殊性，我们在实现语法时将mainclass单独了出来，便于之后利用语法错误来检测是否实现了main class。

-b.switch中break的检测

按照［1］文档中提到的，Java中使用switch每个case中可以不写break，而继续执行下去，直到遇到“break”，但是这往往引起程序员意外的错误。故在本项目中加入了对case语句的检测，每个“case”必须由“break”来结束，否则通不过语法编译。 同时，由于break一定存在，所以ast中可以省略。




-c.float/String的识别

我们认为float/String等还是比较常用的（比如入门的Helloworld程序），故也加入了基础类型识别以及对应的数组识别。

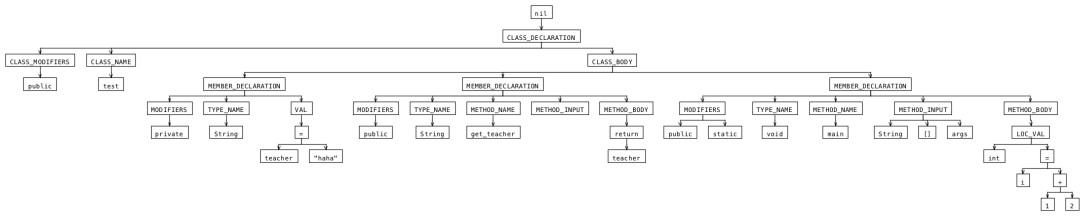
## 五.实验结果：

### 1.cst与ast比较：

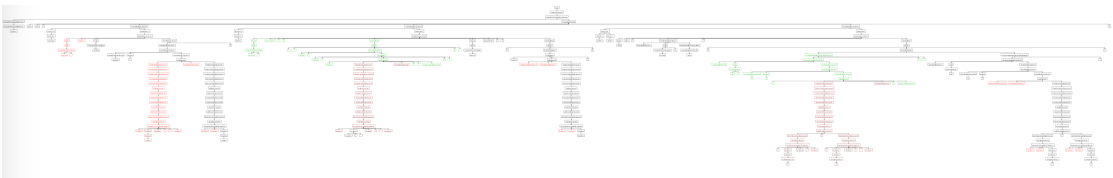
以下为test.java

```
public class test {
	public static void main(String[] args) {
		System.out.print("HelloWorld!");
		return 0;
	}
}
```

同样的代码我们设想的ast相比cst简化很多，也抽象很多。如下2图：




(test.java的ast)



（test.java的cst）

### 2.错误检测演示：

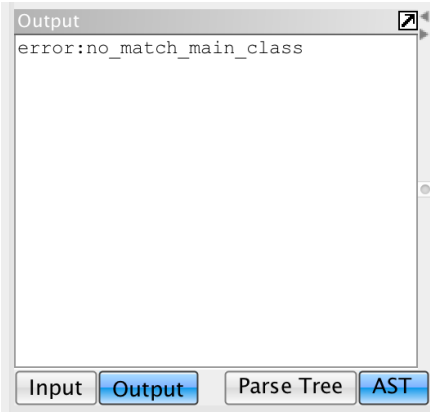
基于antlr3的词法／语法层面检测。（语义错误需要antlr再外接程序，但antlr3使用IDE antlrwork故得到它的输出麻烦，故没有实现。）

-a. 漏写了main class

我们修改了匹配的语法规则。对于错误的情况进行了匹配（所有都是由private / 默认的class组成的），在这种情况下输出”error:no_match_main_class”。正确的话就不会输出。

```
compilationUnit
   :  classOrInterfaceDeclaration* {System.out.println("error:no_match_main_class");}
      | classOrInterfaceDeclaration* mainclass classOrInterfaceDeclaration*
   ;
```

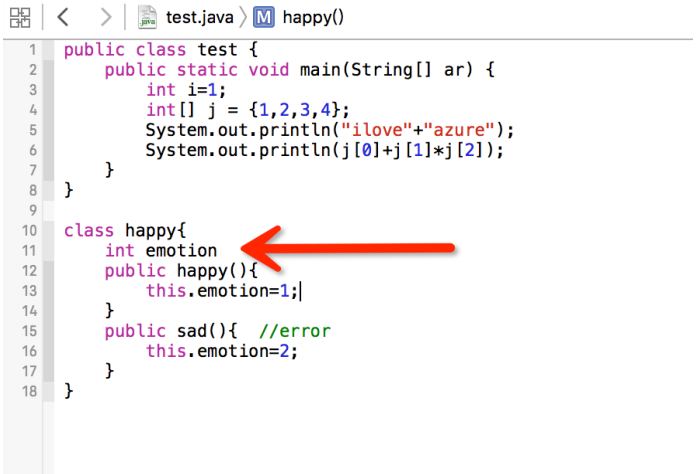
输出在antlrwork中的Output界面




-b. 缺少符号提示以及位置提示

antler3中通过exception来报错，但无具体的错误提示。此处我们使用antlr4，其提供错误的定位。（Minijava.g4在test文件夹中，作为antlr4的语法文件）

比如我在11行去掉一个分号




在终端中就会显示12行无法匹配。

```
➜  mini-java git:(master) ✗ grun Minijava goal -gui <test.java
line 12:4 no viable alternative at input 'intemotionpublic'
```

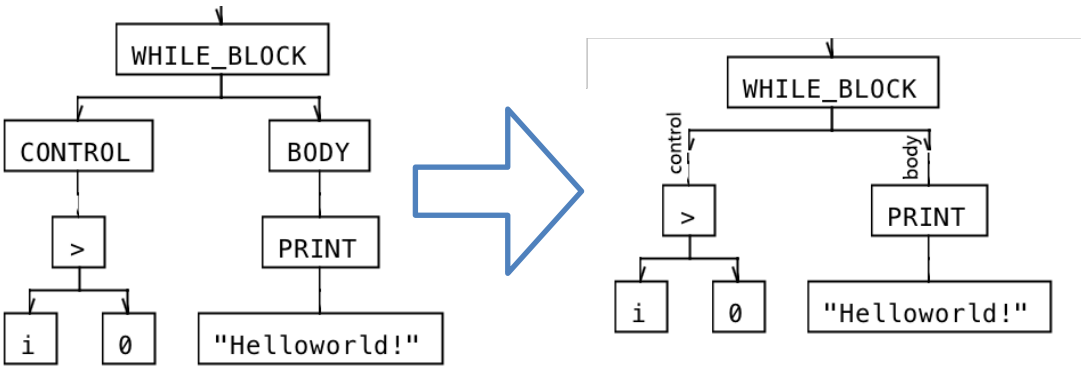
六.实验思考及实验细节补充：

**1.为什么使用了ANTLR3而不是ANTLR4?**

ANTLR4封装了内部代码，直接输出一个cst，我们尝试了修改，读取输出文本，但无法再使用他自身的gui再展示出来，或者就要显式的将语法文件写入代码，这个就可能偏离了使用antlr的初衷。故最后选择了结构相对更灵活的ANTLR3，参考了［2］的例子和说明。将cst变为ast。

**2.关于AST树结构的思考**

实验中的有些信息放在树的线上可能更合适。不过，在本项目实现中，为了显示的清晰，就单独拿出来做节点了。比如while　block中分为control的条件和body，如果将这两个token放在树的边上，ast会更简洁。




**3.使用ANTLR的体会**

ANTLR使用起来还是很方便的。v3中有debug选项，可以一步步看到每个词解析的过程，虽然知道解析的原理，但是看一个程序慢慢出来一个树结构，感觉很厉害。v4中会有直接生成的gui版cst，其实也比较简洁。

引用：

［1］：An Overview of MiniJava EricRoberts: http://cs.stanford.edu/people/eroberts/papers/SIGCSE-2001/MiniJava.pdf

［2］：ANTLR3-Tree-Construction-Webpage: https://theantlrguy.atlassian.net/wiki/display/ANTLR3/Tree+construction