编译原理实验报告

14307130203 林俊雄 13307130428 朱天灵

1.1 项目背景和工具选择

本次课设的主要工作是为 miniJava 语言构造一个编译器的前端，将输入的 miniJava 语言代码段转化成抽象语法树.选用的工具为 antlr4.它是一个可以根据输入自动生成语法树并可视化的显示出来的开源语法分析器.集成了词法分析器和语法分析器.可自动生成 java 代码文件.

它的语法分析器是一个 LL(1)分析器，即采用递归下降分析,预测是每次最多向后读取一个词法元素.

1.2 项目内容:

我们实现的语法完全兼容 MiniJava,即符合 MiniJava 语法的文件一定可以在我们的编译器中编译通过.此外我们还进行了如下扩充和改进:

(1)类的数据成员和函数成员前面可以加 public、private、default、protected 和 static 关键字来声明成员的存取权限和作用域

(2)补充了常见的比较运算符和逻辑运算符

(3)各个类的书写顺序没有要求

(4)不强制一定有一个类含有主函数

(5)增加了 return 语句

(6)支持多维数组的声明和元素访问.

1.3 实验步骤

一、 实验准备

1、新建一个 Submit 目录,以后的操作都在本目录下进行。

2、编写 MiniJava 的语法文件 MiniJava.g4，该文件的部分内容如下所示

```
grammar MiniJava;

/** The start rule; begin parsing here. */
prog:   main_class   ( class_decl )*
    ;

main_class  :   'class' ID '{'
        'public' 'static' 'void' 'main'
        '(' 'String' '[' ']' ID ')'
        '{' stat '}'
        '}'
        ;
stat:
    '{' ( stat )* '}'
    |   'if' '(' expr ')' stat 'else' stat
    |   'while' '(' expr ')' stat
    |   'System.out.println' '(' expr ')' ';'
    |   ID '=' expr ';'
    |   ID '[' expr ']' '=' expr ';'
    ;
```

3、下载 antlr-4.7.1-complete.jar

4、仿照课本（Pragmatic.The Definitive ANTLR 4 Reference.2013）里 1.1 的例子，编写一个自动编译文件文件脚本 antlr4.bat，内容如下：

```
java -cp .\antlr-4.7.1-complete.jar;%CLASSPATH% org.antlr.v4.Tool %*
```

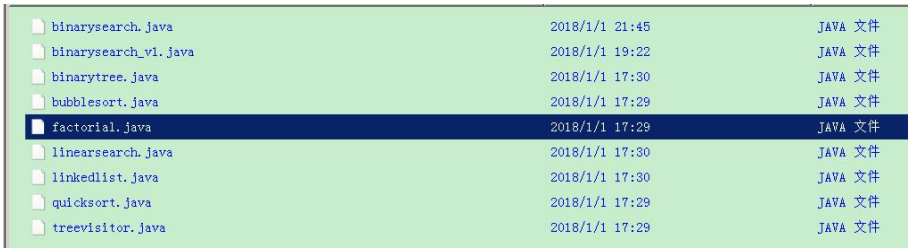
5、仿照课本里 1.1 的例子，编写一个自动运行 TestRig 的脚本，名字为 grun.bat 内容如下：

```
java -cp .\antlr-4.7.1-complete.jar;%CLASSPATH%
org.antlr.v4.gui.TestRig MiniJava prog -gui TestCase/%1.java
```

6、再编写一个用于自动完整编译和测试的脚本 autotest.bat 内容如下

```
del *.java
call antlr4.bat MiniJava.g4
javac -cp .\antlr-4.7.1-complete.jar *.java
call grun.bat %1
```

7、从 MiniJava 的网站上下载测试用的 java 代码文件，一共 9 个，把它们存到 TestCase 子目录下.如下图所示



二、实验步骤

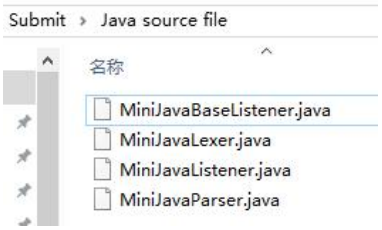
1. 在命令行进入 submit 目录.如下图所示

```
C:\Users\庆领>cd desktop

C:\Users\庆领\Desktop>cd submit

C:\Users\庆领\Desktop\Submit>
```

2. 执行命令:antlr4 MiniJava.g4 编译文法文件,如下图所示

```
C:\Users\庆领\Desktop\Submit>antlr4 MiniJava.g4

C:\Users\庆领\Desktop\Submit>java -cp .\antlr-4.7.1-complete.jar;.:C:\Java\jdk1.
8.0_92\lib\dt.jar;C:\Java\jdk1.8.0_92\lib\tools.jar; org.antlr.v4.Tool MiniJava.
g4
```

3. 此时会自动在当前目录下面生成 4 个 java 代码文件,如下图所示



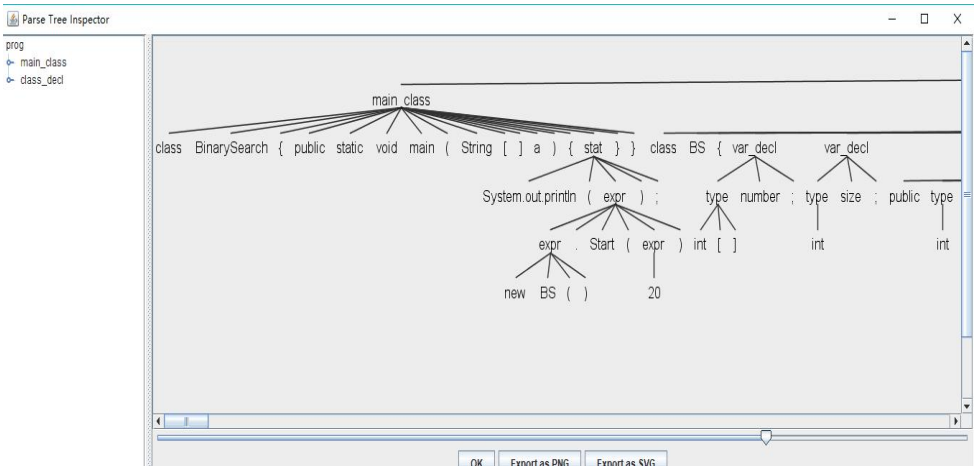
4. 然后执行 javac -cp .\antlr-4.7.1-complete.jar *.java,如下图所示

```
C:\Users\庆领\Desktop\Submit>javac -cp .\antlr-4.7.1-complete.jar *.java

C:\Users\庆领\Desktop\Submit>
```

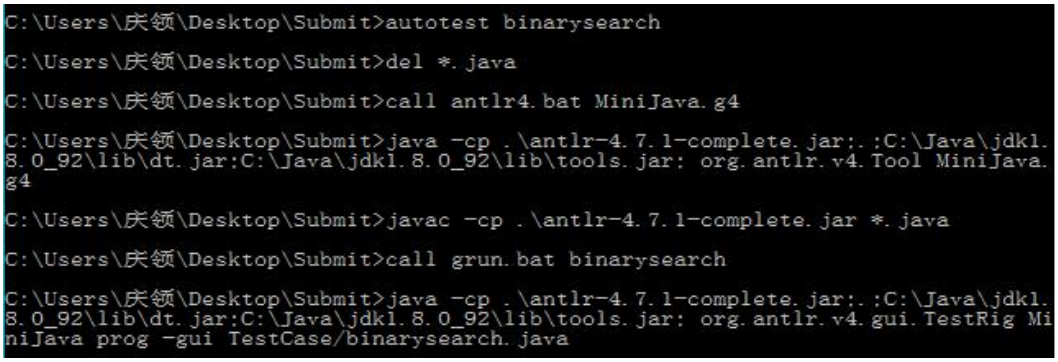
5. 输入命令 grun binarysearch,如下图所示

```
C:\Users\庆领\Desktop\Submit>grun binarysearch

C:\Users\庆领\Desktop\Submit>java -cp .\antlr-4.7.1-complete.jar;.:C:\Java\jdk1.
8.0_92\lib\dt.jar;C:\Java\jdk1.8.0_92\lib\tools.jar; org.antlr.v4.gui.TestRig Mi
niJava prog -gui TestCase/binarysearch.java
```

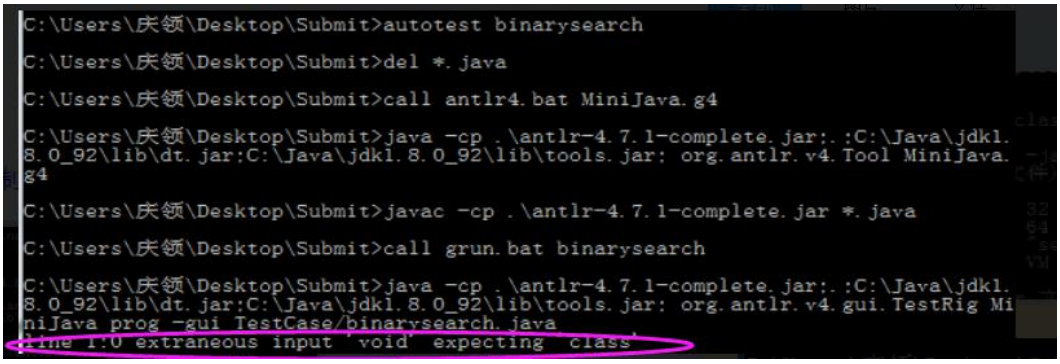
生成的语法树如下所示



6. 前面是分步测试的完整步骤,当所有代码都测试完毕之后,可以进行完整的测试,即执行 autotest binarysearch,效果和前面 6 步完全一样，如果想测试其他文件，则将 binarysearch 改成对应的测试代码文件名即可.



7. 现在我们修改 binarysearch.java 的内容,在最前面加一个单词 void,也就是故意提供一个有错的文件,然后自动测试，此时就会报错,如下图画圈处所示



1.4 代码思路说明

我们自己书写的主要代码都在 MiniJava.g4 这个文件中,该文件的内容主要分为如下几部分：

1、导入必要的 java 工具包

2、一些辅助语法分析的工具函数.主要功能有

(1)检查某个 ID 是否和关键字重复

(2)检查某个变量的类型名是否符合要求

(3)检查二元运算符的两个操作数的类型是否符合要求

(4)判断某个类名是否已经存在

(5)判断某个函数是否已经在本类中定义,用于实现函数重载机制

3、规则语法部分.每个规则可以有动作部分,包含在一对大括号中.且可以带属性声明和参数声明和返回值声明,写在一对中括号里,并用关键字 locals 和 returns 指出.

4、简单的终结符定义,比如运算符、变量名和空白等

代码的主要实现要点是:

1、每个 prog 对象维护一个 class_list 列表,这个列表存储了程序中的所有类的信息

2、每个类对象 Class_declContext 中存储了两个列表 vars_list 和 funcs_list,前者为成员数据列表,后者为成员函数列表

3、每个函数对象 Method_declContext 中存储了两个列表 vars_list 和 para_list。前者为局部变量列表,后者为函数参数列表

4、每个表达式对象 ExprContext 都存储一个 type 字段,用来保存表达的类型名,可用于类型检测.

1.5 错误处理和修复

antlr4 生成的 java 代码自带错误修复功能,主要体现如果遇到错误的词法元素或者语法元素时，程序不是立刻退出，而是尽量忽略当前输入，向后遍历，找到第一个符合预期的输入元素,保证了系统的稳定性.在此基础上,我们又加入了一些检测语义错误的代码,总体来说,本系统目前可以处理如下错误并继续正常运行

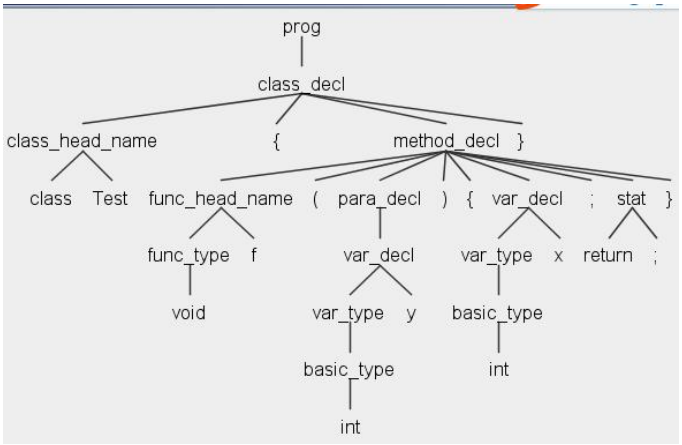
1. 词法错误,如果非法字符比如#，则自动忽略，继续寻找下一个合法字符.测试用例文件为 Testcase 目录下的 error_test9.java,该文件的内容如下

```
//测试非法字符错误
#
class Test{
    void f(int y){
        int x;
        return;
    }
}
```

执行 autotest error_test9 命令之后,控制台输出如下

```
line 2:0 token recognition error at: '#'
add new class name:Test to class name list
add new par: y in function f successful
add new var: x in function f successful
add function:f to class Test success
```

输出的语法树如下:



2. 用关键字做 ID 的,测试文件为为 Testcase 目录下的 error_test9.java,该文件的内

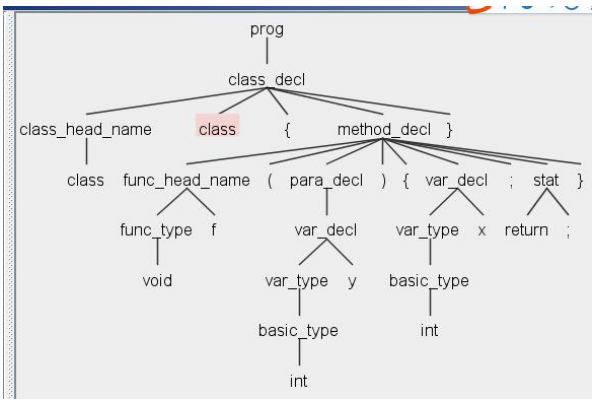
容如下

```
class class{
    void f(int y){
        int x;
        return;
    }
}
```

执行 autotest error_test9 命令之后,控制台输出如下

```
line 2:6 mismatched input 'class' expecting ID
add new par: y in function f successful
add new var: x in function f successful
add function:f to class null success
```

语法树如下



3. 类名重复错误,测试文件是 error_test1, 文件内容如下:

```
//测试重名类错误
class Test{

}
class Test{

}
```

测试效果如下

```
.TestRig MiniJava prog -gui TestCase/error_test1.java
add new class name:Test to class name list
class name: Test is refined at line 5:6
```

红线处信息提示类被重新定义了

4. 类变量重名错误,测试文件是 error_test2, 文件内容如下:

```
//测试重名成员变量错误
class Test{
    int x;
    boolean x;
}
```

测试效果如下

```
add new class name:Test to class name list
add new member var: x in class Test successful
var x is refined at line 4:9
```

红线处信息提示类被重新定义了

5. 参数重名错误.测试文件是 error_test3.java，文件内容如下：

```
//测试重名参数变量错误
class Test{
    int f(){
    }
    int g(int x,int x){

    }
}
```

执行命令 autotest error_test3 之后,命令行输出如下

```
add new class name:Test to class name list
add function:f to class Test success
add new par: x in function g successful
para x in functiong is refined at line 5:17
add function:g to class Test success
```

红线处信息提示参数 x 重名了

6. 函数变量和参数重名错误，测试文件是 error_test4.java，文件内容如下：

```
//测试重名变量错误
class Test{
    int g(int a){
        int a;
    }
}
```

执行命令 autotest error_test4 之后,命令行输出如下

```
add new class name:Test to class name list
add new par: a in function g successful
var a in function g has same name with some para at line 4:6
add function:g to class Test success
```

红线处信息提示变量 a 和某个参数重名了

7. 函数内重名变量错误，测试文件是 error_test5.java，文件内容如下：

```
//测试重名变量错误
class Test{
    int x;
    int f(int x){
        int y;
        int z;

    }
    int g(int a){
        int b;
        boolean b;
    }
}
```

执行命令 autotest error_test5 之后,命令行输出如下

```
add new class name:Test to class name list
add new member var: x in class Test successful
add new par: x in function f successful
add new var: y in function f successful
add new var: z in function f successful
add function:f to class Test success
add new par: a in function g successful
add new var: b in function g successful
var b in functiong is refined at line 11:10
add function:g to class Test success
```

红线处信息提示变量 b 被重新定义了.

8. 函数重载错误,即两个函数如果名字和参数列表完全一样的话,则有冲突,比如在下面的测试文件 error_test6.java 中,函数 f 的重载是正确的,而函数 g 则引发了冲突,因为具有二义性.

```
//测试重名函数错误
class Test{
    void f(int x){  }
    void f(int y,int x){    }
    void g(int x){  }
    void g(int y){}
}
```

执行命令 autotest error_test6 之后,命令行输出如下

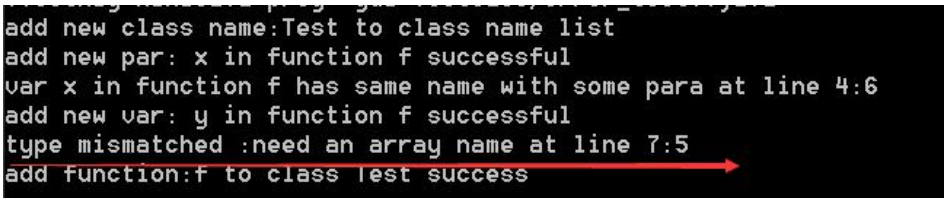
```
add new class name:Test to class name list
add new par: x in function f successful
add function:f to class Test success
add new par: y in function f successful
add new par: x in function f successful
add function:f to class Test success
add new par: x in function g successful
add function:g to class Test success
add new par: y in function g successful
overload function:g in class Testfailed at line 6:7
```

红线部分说 g 的重载失败

9. 数组元素表达式,如果第一操作数不是数组类型时,则报错.测试文件 error_test7.java 的内容如下:

```
1  //测试算术表达式的类型错误
2  class Test{
3      void f(int x){
4          int x;
5          int[] y;
6          x=y[2];
7          x=x[2];
8          return;
9      }
10 }
```

执行命令 autotest error_test5 之后,命令行输出如下

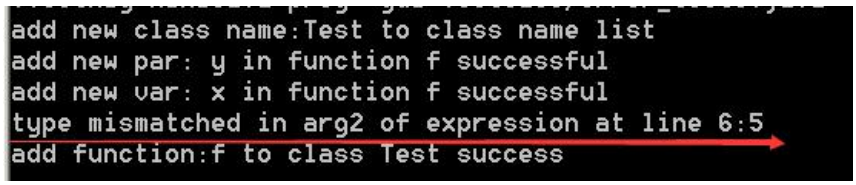


红线处的信息意思就是需要一个数组名.

10. 算术表达式的操作数必须都是整数类型,否则报错. 测试文件 error_test8.java 的内容如下:

```
1 //测试算术表达式的类型错误
2 class Test{
3     void f(int y){
4         int x;
5         x=2+3; //正确
6         x=2+true;  //错误
7         return;
8     }
9 }
```

执行命令 autotest error_test8 之后,命令行输出如下



红线处信息表示第 6 行的表达式的第 2 个操作数类型有误.

1.6 遇到的问题及解决方案

1. 处理赋值语句时,最开始的规则写法是

```
stat: ID ASSIGN_OP expr ';'
    |  ID '[' expr ']' '=' expr ';'
```

测试时在函数里写如下语句:

```
x =2;
```

结果语法分析器总在这里报错.经过思考和调试,发现原因是 antlr 是 LL(1)的,每次最多向后读取一个词法元素,而开始的写法右边的两条展开式首部相同,造成 parser 无法预测采用哪个分支.改进之后为如下形式,测试通过

```
stat:  ID (array_index)? ASSIGN_OP expr ';'
array_index:
       ('[' expr ']')+;
```

这样就使赋值号的左边即可以是一个简单的变量也可以是一个数组元素.

2. 语义分析需要记录每个 ID 的类型名和作用域等信息,开始由于对 antlr 使用不熟练,

采用的是手工修改生成的类代码文件,在里面加入对应成员.经过深入学习后,发现antlr自带了属性机制,即只要在语法规则的locals选项里加入属性名即可,实例如下所示:

```
var_decl[boolean static_flag,MiniJavaParser.Scope var_scope]
locals
[
   String type;
   String name;
   boolean is_static = false;
   MiniJavaParser.Scope scope
]
:var_type ID
```

3. 查找变量时,如果在本作用域没找到，需要继续到父作用域查找,我开始用的思路是增加一个全局栈,存储遇到的每个规则节点,查找时,把每个规则对象依次出栈,在该对象里查找符号.后来发现antlr中提供了getParent函数可以直接访问父对象,大大减少了代码量.

1.7 感想和展望

通过本次课设,我对词法分析、语法分析、语义分析器的设计和实现有了更直观和深入的理解,也掌握了antlr4的使用技巧,深刻的体会到,语法的设计和编译器的实现是一件非常复杂和精巧的项目,必须通过动手实践才能深刻体会到背后的思想和理论.

限于时间和本人的能力，我实现的编译器还有很大的改进余地,比如可以增加对赋值语句两侧的表达式的类型一致性的检查,检查某个对象是否存在,某个函数是否存在,返回语句的类型是否和函数声明类型一致等等,希望能在以后的课程中对它加入完善和改进.