

# MiniJava编译器

## ——编译课程项目报告

**课 程： 编译**

**学 号 1： 15307130416**

**姓 名 1： 韦霞杰**

**学 号 2： 15307130312**

**姓 名 2： 曹 楠**

## 分工说明

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **分工内容** |
| 曹楠 | 15307130312 | 错误类型及错误测试、撰写报告 |
| 韦霞杰 | 15307130416 | 撰写语法分析文件、撰写报告、课程演示 |

**目录**

**1. 实验概述4**

**2. 原理介绍 4**

2.1 编译原理 4

2.2语言识别工具：ANTLR4

2.3 Minijava语言 5

2.4 代码思路 6

**3. 编写文法文件7**

3.1 词法分析 7

3.2 语法分析7

3.3 抽象语法树7

3.4 扩展功能8

**4. 编译器使用步骤8**

**5. 错误处理9**

**6. 分析与感想13**

1. **实验概述**

本次实验目的是为 miniJava 语言（BNF、详细定义及样例）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。使用词法/语法自动生成工具ANTLR得到语法/词法分析器，再添加Java代码使其输出正确的抽象语法树并完善错误处理功能。本项目注重实现编译器的前端，识别 MiniJava 语⾔， 以抽象语法树为输出，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，最后还实现了错误检查。

1. **原理介绍**
   1. **编译原理**

编译是利用编译程序从源语⾔程序转换成目标语⾔的⼀个过程。 其主要包括前端和后端两个部分，前端用于将源语⾔程序翻译成中间表示树(IR 树)，主要包括了词法分析、语法分析、语义分析、栈帧布局等动作，后端用于将中间表示树转译成目标语⾔代码(常常为汇编语⾔)，包括了规范化、指令选择、控制流分析、数据流分析、寄存器分配等操作。

词法分析，主要是用于对程序中每⼀个词的识别，例如 ID, 数字，保留字等。词法分析可以通过正则表达式来表示，用有限状态自动机实现。

语法分析，主要是对程序中每句话语法的判别，通常采用上下⽂⽆关⽂法，利用下推自动机进⾏实现。

语义分析，则是判断程序的语义正确性，包括了各类变量是否已经定义过、变量类型是否匹配、⽅法的参数列表是否匹配等等。

通过这⼏步之后，我们可以得到源程序的⼀棵抽象语法树。抽象语法树将源代码表示成树的形式，以便于后面的处理

* 1. **语言识别工具：ANTLR**

ANTLR (ANother Tool for Language Recognition ) 是一种语言工具，它提供了一个框架，可以通过包含 Java, C++, 或 C# 动作（action）的语法描述来构造语言识别器，编译器和解释器。Antlr 使用上下文无关文法描述语言。在 Antlr 中通过解析用户自定义的上下文无关文法，自动生成词法分析器 (Lexer)、语法分析器 (Parser) 和树分析器 (Tree Parser)。使用 Anltr 的语法可以定义目标语言的词法记号和语法规则，Antlr 自动生成目标语言的词法分析器和语法分析器；此外，如果在语法规则中指定抽象语法树的规则，在生成语法分析器的同时，Antlr 还能够生成抽象语法树；最终使用树分析器遍历抽象语法树，完成语义分析和中间代码生成。

ANTLR 的⽂法定义主要分两部分，parser rule和lexer rule。区分这两种规则的⽅式是⼤小写，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，匹配第⼀个遇到的合法规则。同时ANTLR还有如下优点：

（1）有自⼰的主页和详细的⽂档，上⼿容易。

（2）有相应版本的参考书籍。

 （3）⽂法定义友好，输出⽅式多样化。

* 1. **Minijava语言**

Mini Java是Java语言的子集，它与Java相比做了如下精简：

（1）不允许重载。

（2）一个文件中可以申明若干个类，但必须有且只有一个的主类，辅类可以有多个，类不能申明为public；主类中只能有一个主方法，该方法的签名必须为“public static void main(String[])”，其中String[]参数不做处理。主方法中只能有一条输出语句：System.out.println( int )，该语句只能输出整型变量值。

（3）只有类，没有接口，有继承关系（单继承）。

（4）类中只能申明变量和方法。

（5）只有四种变量类型：整型（integer）、布尔型（boolean）、数组（array）、对象；只有一类数组：整型数组（int[]）；变量必须申明为(auto)型。

（6）方法必须为public，必须有返回值，返回值类型受（5）限制；可以有参数，可以没有，参数数量没有限制，类型受（5）限制

（7）一共有6中语句（statement）：代码块（block）、简单赋值语句（assignment statement）、数组赋值语句（array assignment statement）、打印语句（print statement）、if语句、while语句。

（8）一共有9种表达式（expression）：与（and）、比较（compare）、加（plus）、减（minus）、乘（times）、数组定位（array lookup）、数组长度（array length）、消息传递（message sending，即参数传递）、主表达式（primary expression）；与表达式为“短路与”（&&）；比较只能作小于比较。

（9）主表达式一共有9种：整数（Integer）、“真”（true）、“假”（false）、对象、this、初始化（allocation）、数组初始化（array allocation）、非（not）、括号（bracket）。

（10）初始化时只能使用空参数，所以没有“构造器（constructor）”概念。（11）对标识符（IDENTIFIER）没有作明确定义，在这里规定只能是字母、数字的集合，但必须以字母开头，区分大小写。

（12）对整数（INTEGER\_LITERAL）没有作明确定义，在这里规定为无符号整数，32位（0～4294967295）。

（13）不允许注释。

（14）类变量声明时不能初始化，初始化必须在方法中完成。

1. 不能定义内部类。
   1. **代码思路**
2. 导入必要的 java 工具包
3. 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

1. 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.
2. 简单的终结符定义,比如运算符、 变量名和空白等
3. **编写文法文件**

本实验的词法分析和语法分析等都编写在MiniJava.g4文件中，src文件夹下其他文件为ANTLR自动生成或者是测试使用文件。以下是文法文件的编写思路。

* 1. **词法分析**

语言的词法定义包括文法定义用到的 3 个关键字记号：IF、WHILE 和 RETURN，以及其他 4 个记号：变量 ID、数字 NUMBER、换行符 NEWLINE 和空白字符 WS。其中换行符 NEWLINE 和空白字符 WS 不需要作为词法记号向语法分析器传递，使用 skip() 方法将其跳过。

ABTLR不支持左递归文法。如果定义了非 LL 文法，必须对其改写。在 LL 文法识别器中，算符的优先级要通过文法的嵌套定义来体现。优先级低的算符要优先声明，相同优先级的算符可以在同一个规则中描述。比如加减运算的优先级比乘除运算的优先级低，加减法要先于乘除法定义。

* 1. **语法分析**

整个语言由若干函数定义组成。每个函数定义包括函数名、参数列表和函数体。函数体由若干语句构成，包括返回语句、条件语句、循环语句、赋值语句和表达式运算语句等。表达式定义了语言支持的若干运算，语言中的运算能力主要由表达式定义来体现。

* 1. **抽象语法树**

抽象语法树是语言的结构化表示形式，它的每个节点代表源程序中的某种构造。抽象语法树往往忽略语言中的某些细节， 比如括号（）。当括号用于表示计算的优先级时，树形结构已经拥有了优先级信息，可以去掉；当括号用于表示分隔符时，比如函数声明和函数调用，分隔符在抽象语法树中不需要存在，可以去掉。抽象语法树不同于语法分析树，后者代表语法分析器分析的完整过程， 而抽象语法树往往使用最少的信息来表示语言本身的结构。可以在定义文法的同时，使用 Anltr 树构造语法来建立抽象语法树，我们语言中的抽象语法树节点主要有以下几类：

1. 使用占位符作为根节点建立树节点
2. 使用算符作为根节点建树节点
3. 依据语义规则条件的不同，建立不同的树节点
4. 通过文法规则重写建立树节点，去掉相关的分隔符号
   1. **扩展功能**

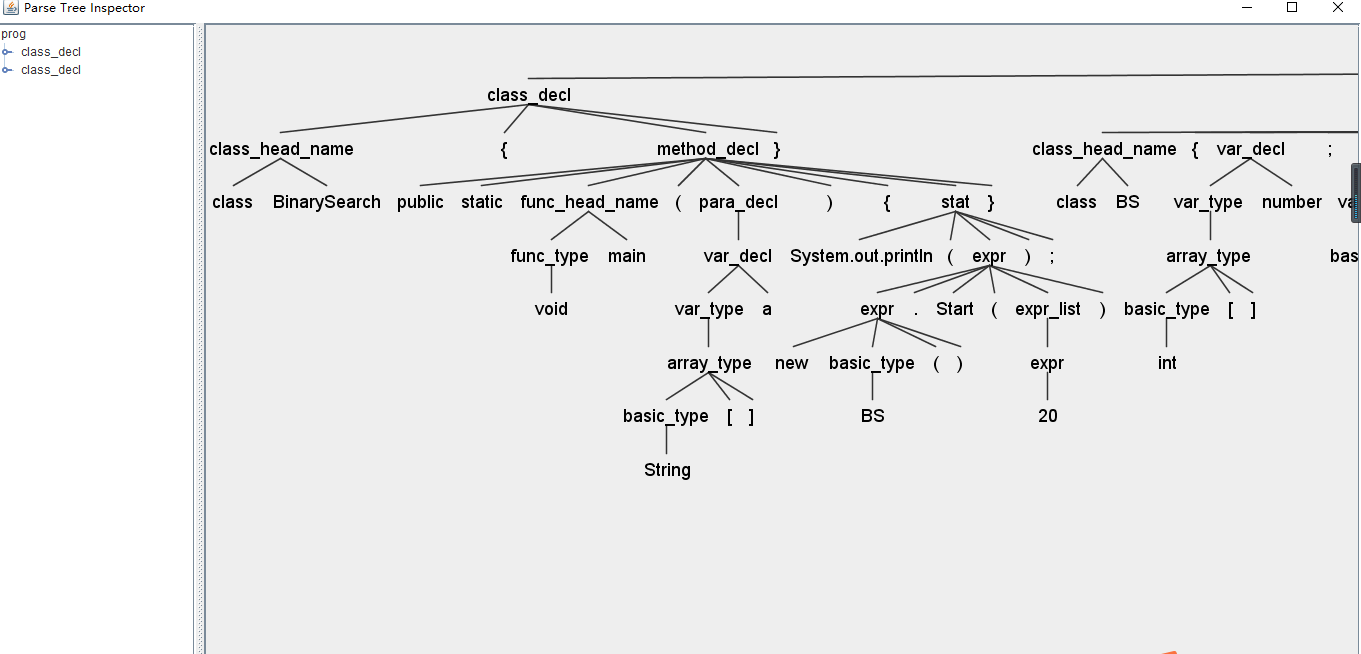
我们对MiniJava语言进行了如下扩展：

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. 允许函数重载
3. 各个类的书写顺序没有要求
4. 增加了 return 语句
5. 不强制一定有一个类含有主函数
6. 补充了常见的比较运算符和逻辑运算符
7. 支持多维数组的声明和元素访问
8. **编译器使用步骤**

（1）编写Minijava.g4语法文件，在其中加入错误处理的java代码

（2）执行antlr4命令：antlr4 MiniJava.g4  
（3）编译生成的java文件，执行命令：javac –cp antlr-4.7.2-complete.jar \*.java（4）添加测试代码binarysearch.java到目录下  
（5）得到语法分析树，执行命令：grun MiniJava prog -gui binarysearch.java

通过以上步骤既可生成binarysearch.java程序的语法分析树，得到下图所示语法分析树。

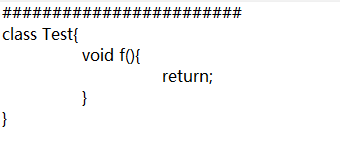


1. **错误处理**

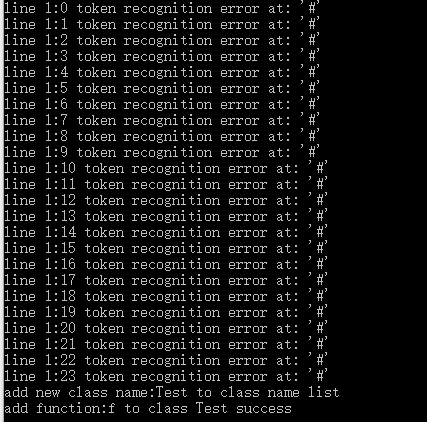
我们在语法文件中添加Java代码控制行为来检测错误，我们可以处理的错误如下。

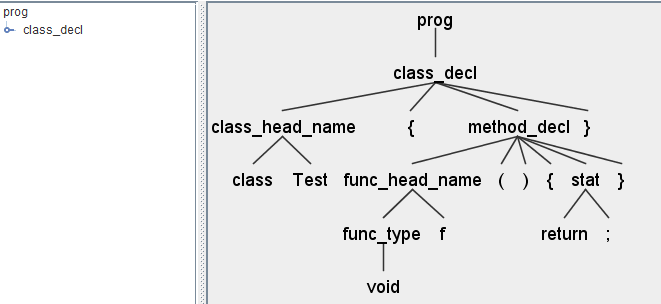
1. 非法字符

文件error\_test1.txt中有一行非法字符，文件代码如下：



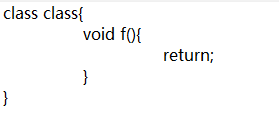
使用MiniJava进行语法分析之后结果如下，正确检测到错误。



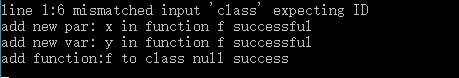


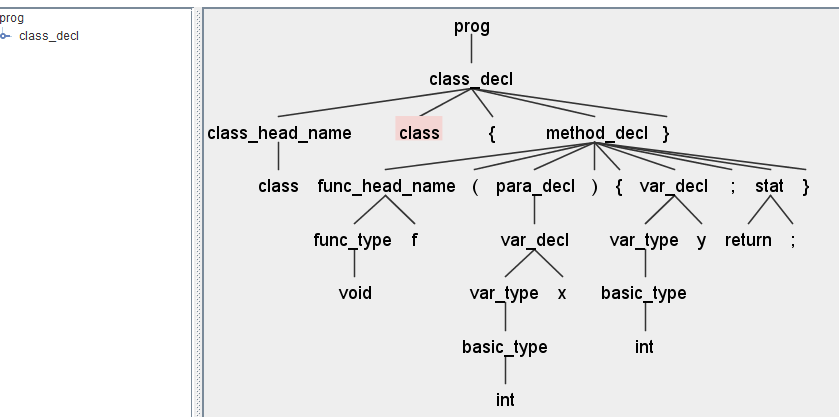
1. ID为关键词

文件error\_test2.txt的代码如下，其中class是关键词。



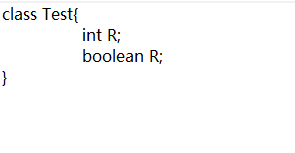
使用MiniJava进行语法分析之后结果如下，抽象语法树上会有红色标记提示出错。



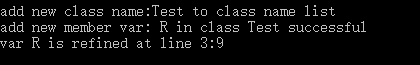


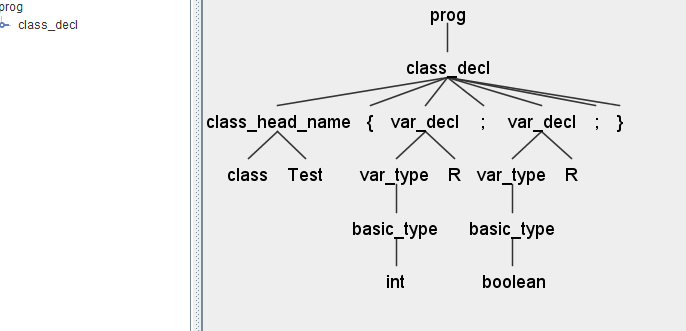
1. 变量重复定义

文件error\_test3的代码如下，变量R被重复定义。



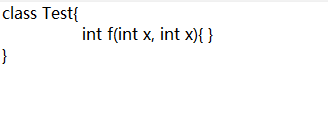
使用MiniJava进行语法分析之后结果如下。



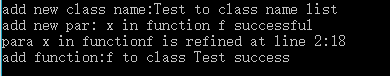


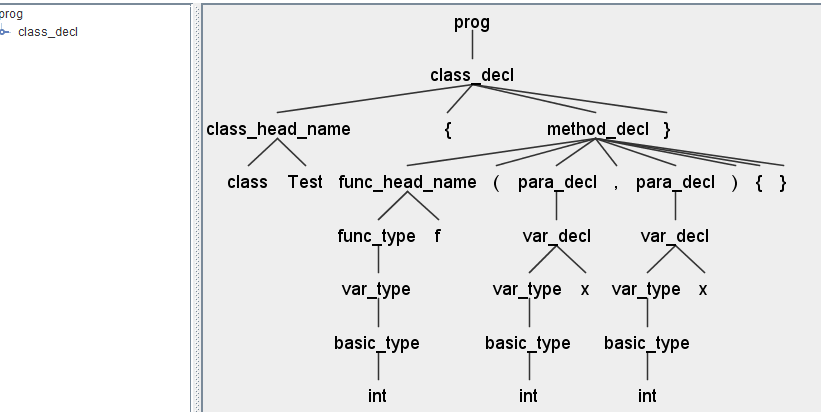
1. 参数重名

文件error\_test4.txt的代码如下，其中函数f()的两个参数同名。



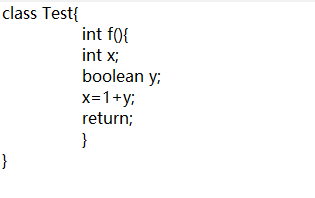
得到结果如下，指出了错误位置和类型。



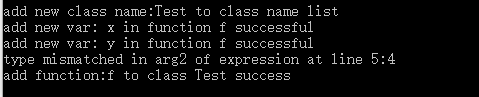


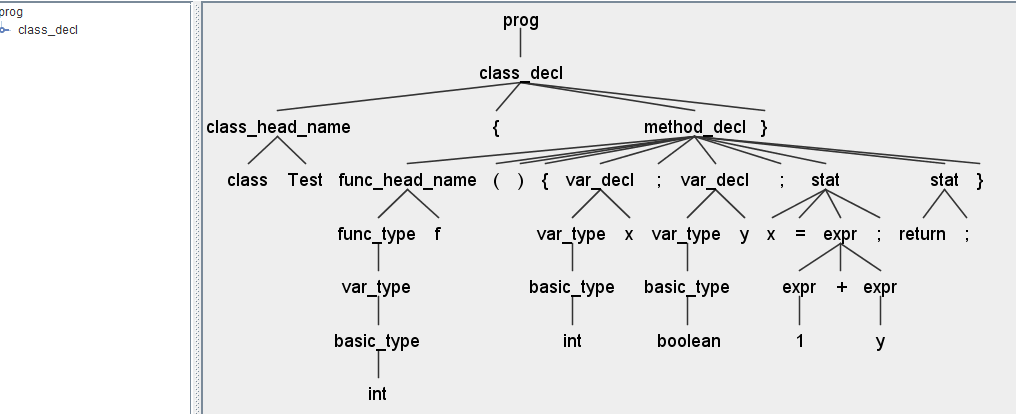
1. 算术表达式类型错误

文件error\_test5.txt的代码如下，其中x和y的类型不能进行加法运算。



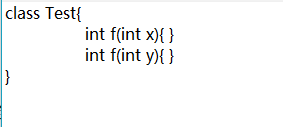
得到结果如下，指出了错误类型和位置。



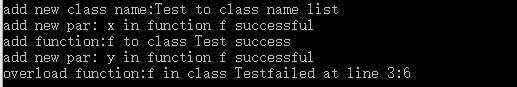


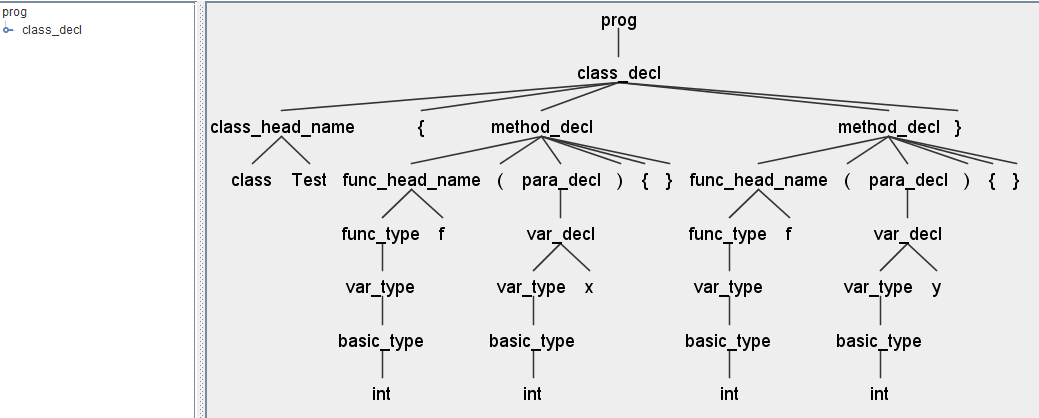
1. 函数重载错误

文件error\_test6.txt的代码如下，其中两个函数重载产生了二义性。



得到结果如下，指出了错误类型和位置。





我们还处理了其他的错误，包括类名重复、类变量重名错误、函数内变量重名错误等，因为处理方式和上面的几个类型差不多，所以不再一一截图分析。

1. **分析与感想**

通过本次项目实现了对编译器前端的编写， 我们加深了对编译原理的理解，了解了编译器实现词法分析、语法分析、语义分析，以及各类错误报告和错误恢复的⼤致流程。同时我们也深入学习ANTLR语法分析器。

整个项目参考了大量ANTLR Reference和网络上的实例。我们在其中也遇到了很多问题，比如处理赋值语句时,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，最后发现原因是 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，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。

stat: ID (array\_index)? ASSIGN\_OP expr ';'

array\_index:

('[' expr ']')+;

在查找变量时，如果在本作用域没找到， 需要继续到父作用域查找。我们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。后来通过参考官方文档发现 antlr 中提供了 getParent 函数可以直接访问父对象，使用之后页可以大大减少代码量。