**编译project——MINIJAVA编译器**

**1: 项目描述**

**1.1: 简介**

*miniJava是一个简化版的Java语言。该项目的目的是为miniJava语言构造一个编译器前端。包括词法分析器，语法分析器，抽象语法树的生成和错误输入的处理功能。这个项目使用了Java语言中最为流行的编译器自动生成框架。通过指定自定义语言的语法，该框架就可辅助生成编译器的前端。*

**1.2: 词法分析器**

*词法分析器又称为Scanner，Lexical analyzer和Tokenizer。程序设计语言通常由关键字和严格定义的语法结构组成。编译的最终目的是将程序设计语言的高层指令翻译成物理机器或虚拟机可以执行的指令。词法分析器的工作是分析量化那些本来毫无意义的字符流，将他们翻译成离散的字符组（也就是一个一个的Token），*

**1.3: 语法分析器 (Parser)**

*编译器又称为Syntactical analyzer。在分析字符流的时候，Lexer不关心所生成的单个Token的语法意义及其与上下文之间的关系，而这就是Parser的工作。语法分析器将收到的Tokens组织起来，并转换成为目标语言语法定义所允许的序列。无论是Lexer还是Parser都是一种识别器，Lexer是字符序列识别器而Parser是Token序列识别器。他们在本质上是类似的东西，而只是在分工上有所不同而已。*

*本项目采用*

**1.4: 抽象语法树 (AST)**

*抽象语法树是源代码的抽象语法结构的树状表示，树上的每个节点都表示源代码中的一种结构，这所以说是抽象的，是因为抽象语法树并不会表示出真实语法出现的每一个细节，比如说，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现。抽象语法树采用上下文无文文法。*

**1.5: 树分析器 (Parser)**

*树分析器可以用于对*[*语法分析*](http://baike.baidu.com/view/487037.htm)*生成的*[*抽象语法树*](http://baike.baidu.com/view/1874022.htm)*进行遍历，并能执行一些相关的操作。*

**2.ANTLR**

**2.1 背景**

*ANTLR 语言识别的一个工具 (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。 计算机语言的解析已经变成了一种非常普遍的工作，在这方面的理论和工具经过近 40 年的发展已经相当成熟，使用 Antlr 等识别工具来识别，解析，构造编译器比手工编程更加容易，同时开发的程序也更易于维护。*

*和大多数语言识别工具一样，Antlr 使用上下文无关文法描述语言。最新的 Antlr 是一个基于 LL(\*) 的语言识别器。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)。*

**2.2 功能**

*识别和处理编程语言是 Antlr 的首要任务，编程语言的处理是一项繁重复杂的任务，为了简化处理，一般的编译技术都将语言处理工作分为前端和后端两个部分。其中前端包括词法分析、语法分析、语义分析、中间代码生成等若干步骤，后端包括目标代码生成和代码优化等步骤。*

*Antlr 致力于解决编译前端的所有工作。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。整个工作在 Anltr 强大的支持下，将变得非常轻松和愉快。*

**3: 使用ANTLR实现编译器前端**

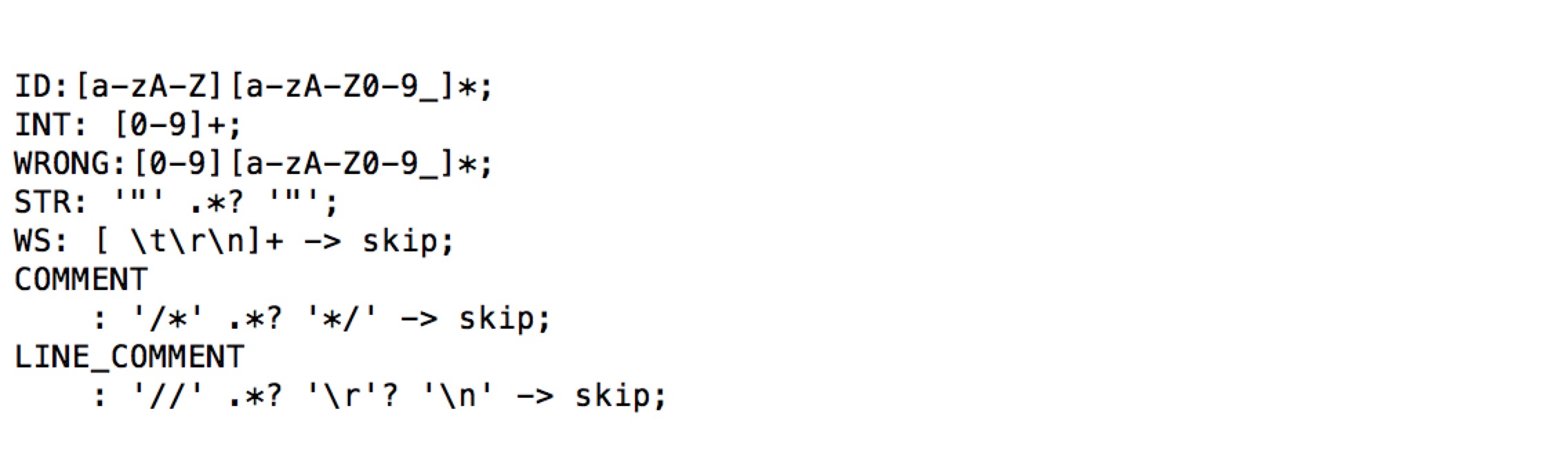
**3.1: 词法定义**

*词法定义利用正则表达式定义了每种词的词法，正则表达式中的标记包括：*

*\*：匹配前面的子表达式任意次*

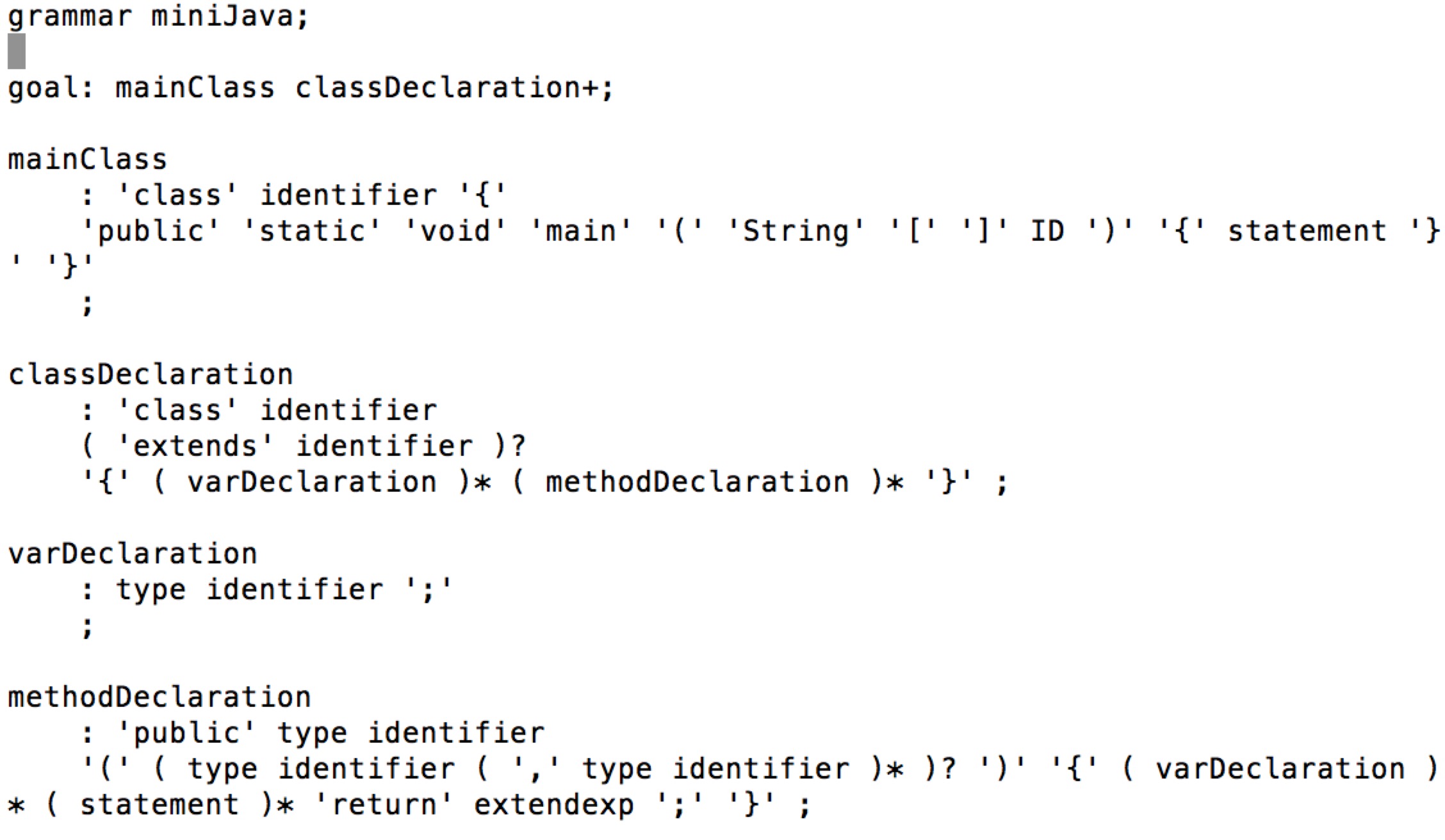
*+：匹配前面的子表达式一次或多次(大于等于1次)*

*?：匹配前面的子表达式零次或一次*

****

**3.2: 语法定义**

*语法定义中定义了数个规则，每个规则递归的调用其他的规则*

**

**3.3: 错误处理**

*ANTLR自带了数个错误处理功能，如提示多余或缺少某个符号，词法匹配不正确等。我们还添加了数个错误处理功能。包括对表达式的括号不匹配的处理，对以数字开头的ID的错误处理。*

**3.4：测试样例**

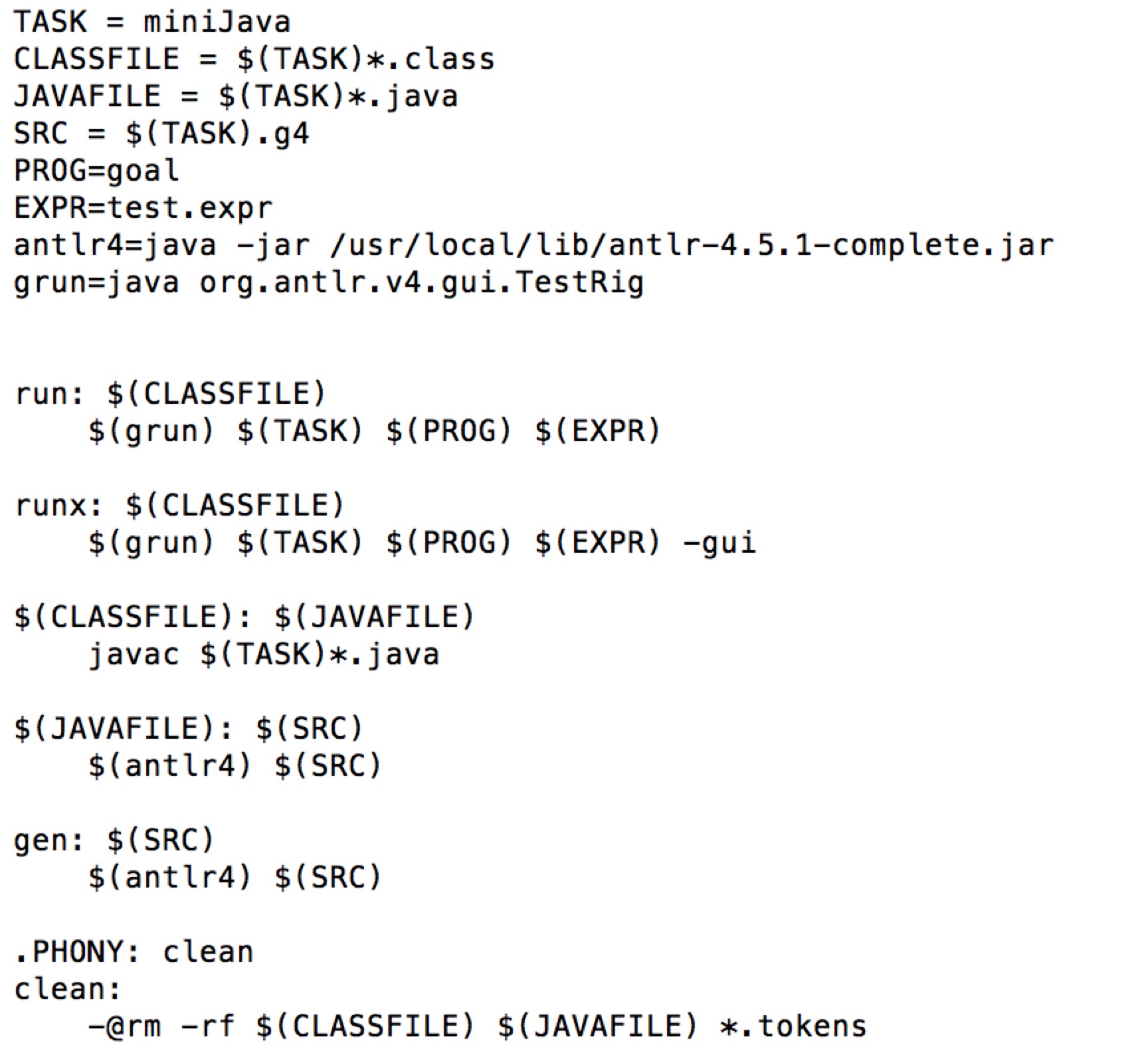
*我们构造一个二叉树的类来测试miniJava语言，二叉树是最常用的数据结构之一，因此使用它来测试我们的编译器最具有代表性。这个类的方法包括为二叉树新增元素，删除元素，遍历二叉树等等。*

**4: 程序使用方法**

**4.1: 执行**

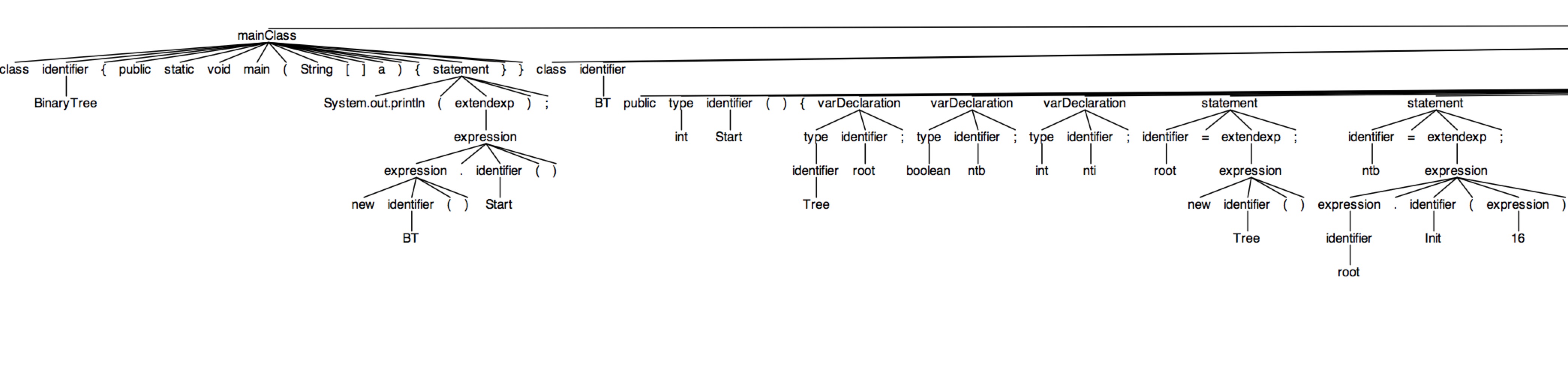
*在项目主目录的命令行调用make run即可执行，使用make runx执行可以显示抽象语法树*

*Makefile的内容如下*

**

**4.2：结果**

*以下是对test.expr中的样例程序进行解析并生成的结果*

**

# 项目体会

做完这个项目，收获很多。由于之前并没有使用过词法分析器和语法分析器的经验，所以学习对于工具的使用也消耗了不少的时间。之后就是关于文法的处理了。这个工作非常需要思维。因为多次的规约会使得很多问题变的复杂起来。在不断解决问题的过程中对于文法中二义性的处理，左递归的处理等问题都有了更深刻的理解。 另外本次实验中第一次接触到github，一开始对github的使用方法和功能都没有任何了解，做完程序以后，才花了半天的时间去建立起一个完整的项目，过程非常曲折。但使用下来发现确实是一个很好的工具，应该以后也会继续去使用。