

**编译原理课程项目**

**Toy compiler**

**组 员：黄俊豪、彭潇**

**学 号：10185101220、10185101221**

**学 院：软件工程学院**

**专 业：嵌入式系统**

**指导教师：琚小明**

**2021年 1 月**

目录

1.背景与现状分析

2.相关技术介绍（如采用工具做的：LEX, YACC等介绍）

3.功能需求分析（要求的基本功能：词法、语法、语义、符号表、出错处理、基本输入/输出显示；提升功能：中间代码生成、良好输入/输出界面与显示）

4.功能设计（编译功能模块、符号表、出错处理、数据结构等）

* 1. 词法：
     1. 按照自动机DFA来实现词法分析；
     2. 扫描设计；
     3. 构词缓冲设计；
     4. 符号表建立与更新；
     5. 关键字与标识符ID的分离；
     6. 词法返回值
  2. 语法：
     1. 语法分析表存储；
     2. 分析堆栈构建与初始化；
     3. 输出语法树构建；
     4. 采用LL(1)算法的语法分析结构；
     5. 符号表更新；
  3. 语义：
     1. 在语法树的基础上对类型进行语义分析
  4. 代码生成：
     1. 代码生成方法；
     2. 代码输出
  5. 符号表：
     1. 符号表的结构；
     2. 符号表的查询、添加、修改等操作
  6. 出错处理：
     1. 出错信息；
     2. 出错定位

5.功能实现（输入/输出显示、界面效果等）

* 1. 基本的输入、输出界面；
  2. 界面布局与设计（自由发挥）

6.测试（测试用例等）

* 1. 自测用例
  2. 检查用例

7.总结（总结技术、结果等）

8.参考文献

9.工作量占比

1. 背景与现状分析

编译原理即是对高级程序语言进行翻译的一门科学技术, 我们都知道计算机程序由程序语言编写而成, 在早期计算机程序语言发展较为缓慢, 因为计算机存储的数据和执行的程序都是由0、1代码组合而成的, 那么在早期程序员编写计算机程序时必须十分了解计算机的底层指令代码通过将这些微程序指令组合排列从而完成一个特定功能的程序, 这就对程序员的要求非常高了。人们一直在研究如何如何高效的开发计算机程序, 使编程的门槛降低。本次实验我们要亲手设计、编制并调试一个Toy编程语言的编译器，加深对编译器的理解，提高程序设计能力和兴趣，为以后的软件开发工作奠定良好的基础。

1. 相关技术介绍（如采用工具做的：LEX, YACC等介绍）

编译器整体采用java实现；其中词法分析用到了子集构造法完成正则表达式到DFA的转换(构建firstpos,fellowpos集合)然后识别token；语法部分包括(构造LL语法分析表，项目集，first集合，fellow集合,语法分析栈，构建语法树等技术)，语义中需用到构造属性文法，中缀表达式转后缀表达式，构造语法制导翻译，三地址表示的代码生成等技术。

项目展示则为前后端分离的we项目，后端采用SpringBoot等技术，前端则通过vue + codemirror 实现。

1. 功能需求分析（要求的基本功能：词法、语法、语义、符号表、出错处理、基本输入/输出显示；提升功能：中间代码生成、良好输入/输出界面与显示）
   1. 词法：

读入源程序的输入字符，将它们组成词素，生成并输出一个词法单元序列，每个词法单元对应于一个词素。将词法分析器设计为一个子例程，它可以被解析器调用并返回一个四元组(tokentype,attributevalue,linenumber,lineposition)

* 1. 语法：

采用LL(1)方法解析语法，在创建语法分析器之前，需要先写出语法分析动作表。 你的parser应该能打印出语法生成树。

* 1. 语义：

能够完成基本的语法推导翻译，不要求做类型检查。

* 1. 符号表：

符号表应该能记录标识符的名字和数值，并且能够快速的添加，查找和修改。

* 1. 中间代码生成：

中间代码采取三段式的形式，并且能做到一定的优化（尽可能的减少中间变量）。 比如a = (1+2)\*3+(1+2)\*4在生成中间代码的时候，将1+2设为一个变量，并两次 使用

* 1. 出错处理：

1. 词法错误

若在词法分析的过程中碰到错误，会将token标记为ERR，记录错误信息，将错误token的位置及错误信息输出到错误表里，然后继续向下分析

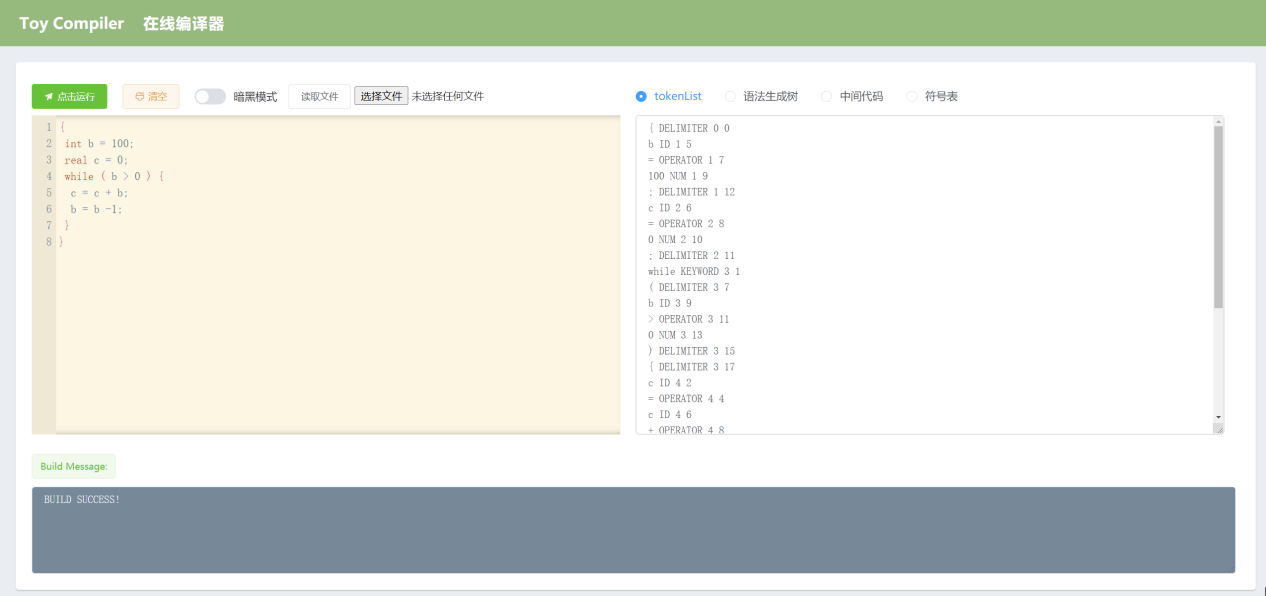
1. 语法错误

若在语法分析的过程中遇见了错误的推导形式，则报错语法解析错误

1. 语义错误

若在语法推导翻译的过程中出现了语义错误，如(除数为0等)，则报语义解析 错误

* 1. 基本输入/输出显示：



输入：对关键字，操作符，分隔符有高亮显示，括号自动补全，同时可以切换主题为暗黑模式，还能读取本地文件内容。





输出：右边可以展示tokenlist，语法生成树，中间代码，符号表。下方会展示build message，提示错误信息。







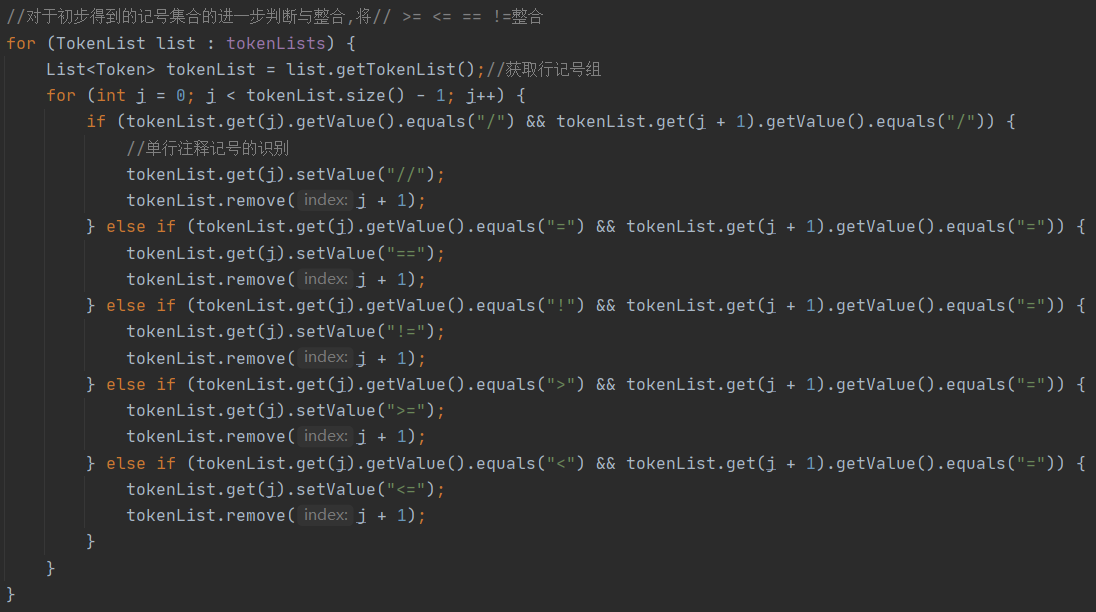


1. 功能设计（编译功能模块、符号表、出错处理、数据结构等）
   1. 词法：
      1. 按照自动机DFA来实现词法分析；

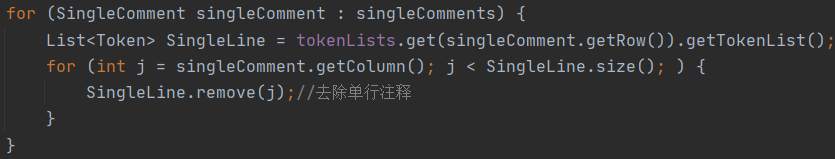
分别定义了分隔符、标识符、整数、实数、操作符、关键字的正则表达式，

* + 1. 扫描设计；

1.函数getTokens读取输入文件到一个字符串，首先，先将这个字符串按照“\n \r”进行划分，然后通过" \n\r\t;(){}\"'+-\*/=<>!"进行分割

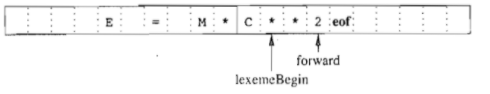
2.对于初步得到的记号集合的进一步判断与整合,将// >= <= == !=整合；

3.第二次对记号进行判断整合，主要用于去除各种分隔符

4.第三次对记号进行去除注释，得到真正的完整的记号

* + 1. 构词缓冲设计；

利用两个交替读入的缓冲区，如下图所示：



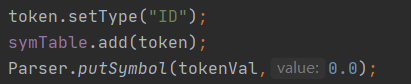
减少用于处理单个输入字符的时间开销，lexemeBegin指针指向当前词素的开始处，当前正试图确定这个词素的结尾；确定一个词素后，lexemeBegin指针指向该词素之后的第一个字符；forward指针一直向前扫描，直到发现某个模式被匹配为止，并且前移时首先要检查是否已经到达某个缓冲区的末尾；确定一个词素后，forward指针将指向该词素结尾的字符。哨兵标记：扩展每个缓冲区，使他们在末尾包含一个"哨兵"字符，就可以把对缓冲区末端的测试和对当前字符的测试合二为一。作用：合并forward指针前移时的检查操作，即检查是否到达缓冲区的末尾和确定读入的字符是什么。哨兵字符必须是一个不会在源程序中出现的特殊字符，可以选择eof，任何不是出现在某个缓冲区末尾的eof都表示达到了输入末尾

* + 1. 符号表建立与更新；

进入词法分析会初始化一个符号表：



在判断token为标识符后会将这个token放进符号表，并且初始化它的值：



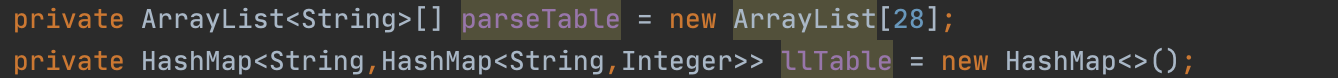
* + 1. 关键字与标识符ID的分离；

对于每一个token首先判断是否是关键字，然后通过标识符的正则表达式判断是否是ID

* + 1. 词法返回值

词法返回值为一个四元组(tokentype,attributevalue,linenumber,lineposition)，四元组中的第一项是令牌类型，第二项是它的属性值，第三项是该令牌所在行的行号，最后一项是该令牌在该行中的位置。

* 1. 语法：
     1. 语法分析表存储；

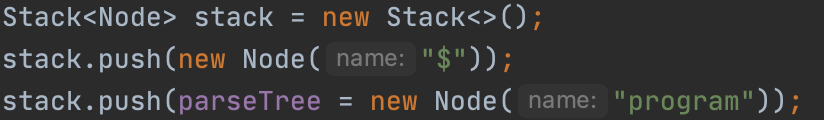


如图所示，用了一个数组parseTable存放了28个推导表达式。

用一个嵌套的hashMap存放语法动作表，（非终结符，终结符）-> 该移进的 推导式的数字。

* + 1. 分析堆栈构建与初始化；

堆栈的初始化，压入了$初始符号。用于判断语法分析的结束。然后放进了program的非终结符。



然后在每一轮循环里判断栈顶的字符和输入字符串的第一个字符的关系。如果都为非终结字符也相等，则弹出栈顶的字符；如果栈顶的字符为非终结符则从语法分析表分析移入的表达式为哪一条，然后将表达式右边的符号全部从右到左全部压入栈中。

若以上情况，都没有满足。则表示语法分析错误，结束分析。



* + 1. 输出语法树构建；

1.首先，构造一个语法树的节点的类Node，如图给出类的属性

name：终结符或者非终结符的名字

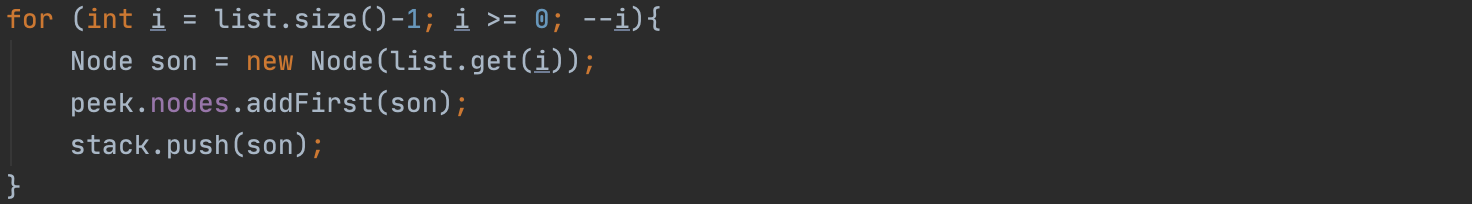
type：终结符(ID,NUM,OPERATOR,DELIMITER...),非终结符(NTERMINAL)

nodes: 孩子节点的链表(用于连接起树中的节点)

和构造方法。



1. 在每次将栈顶的元素弹出的时候，在将其对应的推导表达式右边的符号压入 栈中的同时，也将他们放入弹出节点的子节点中。



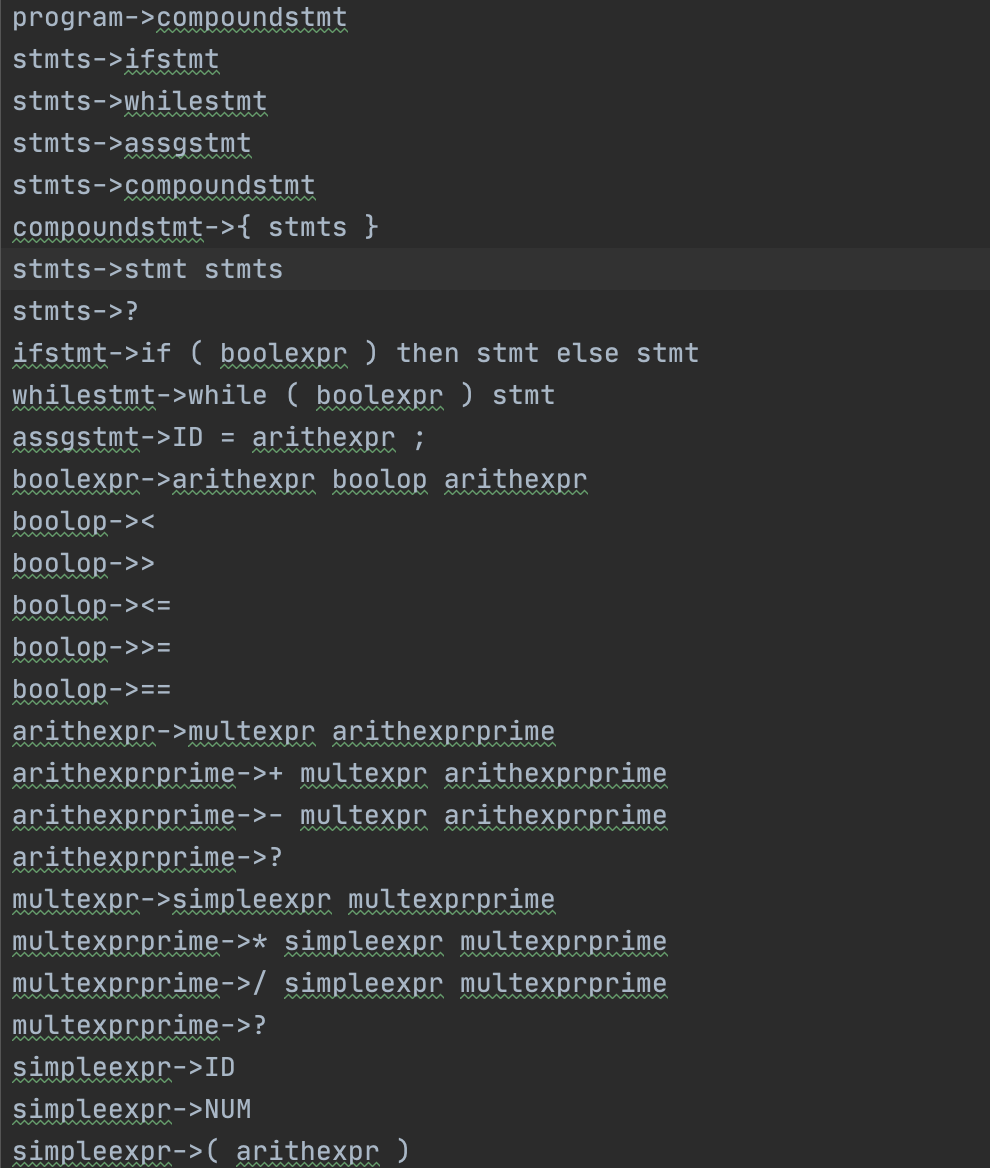
3.至此，当对语法分析进行完成时，一棵语法树也已经建好了。此后只需要通

过先序遍历便可以将语法树打印出来了。

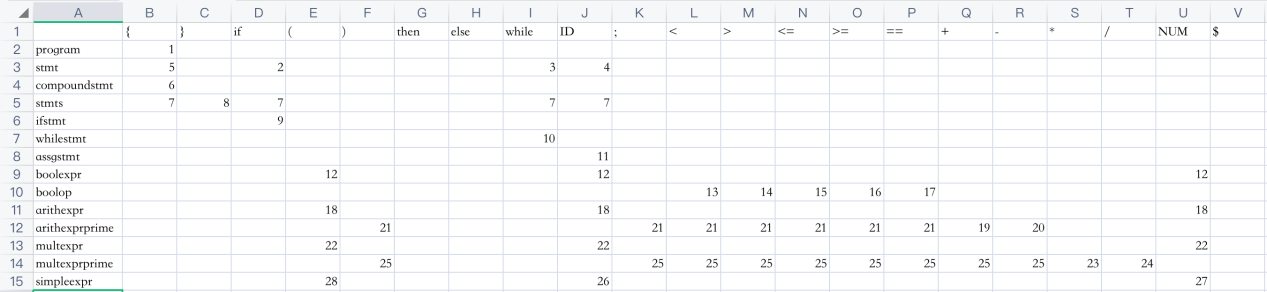
* + 1. 采用LL(1)算法的语法分析结构；

在处理语法推导式的时候，我用了一个数组存下了28个推导式。





之后再手写了该语法的LL(1)分析动作表



其中第A列表格中的字符串表示非终结符，第1列中的字符串表示终结符。

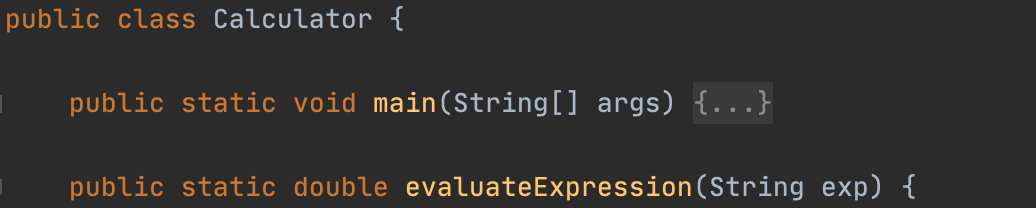
每一个非终结符和终结符对应的数字表示遇到该终结符时该选择弹栈并移进 的推导表达式中右边的符号。

* 1. 语义：
     1. 在语法树的基础上对类型进行语义分析

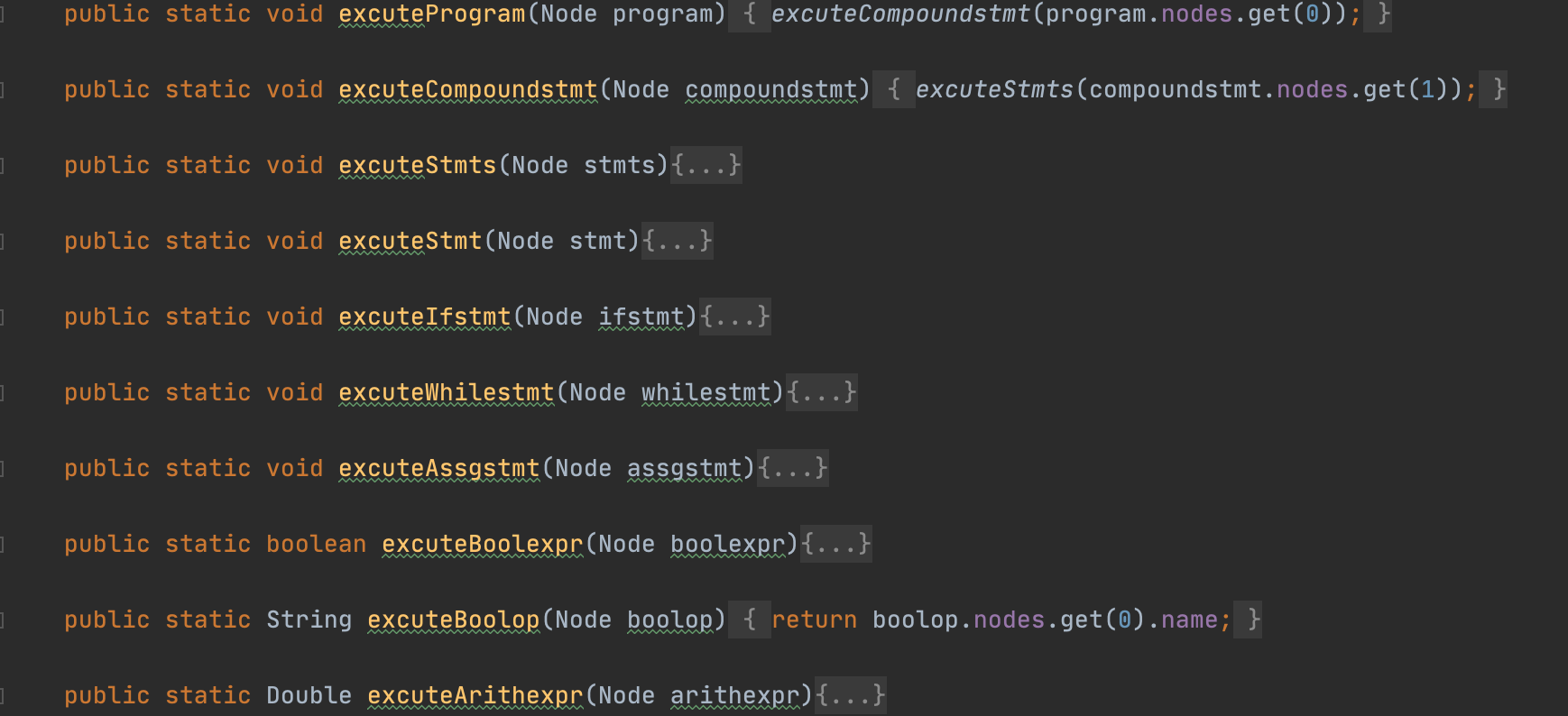
在对语义分析的阶段。因为此文法属于L属性，所以更适合通过LL语法分析 中先生成一棵语法树再加上综合属性和集成属性来完成语义分析。

但是因为此语法比较简单，唯一涉及到需要继承属性的地方可能就是 arithexpr非终结符的推导之处，所以我打算将arithexpr作为基本单元，将其 全部扩展为表达式之后然后通过中缀转后缀表达式进行计算。

如图所示，caculator类中的evaluateExpression用于计算中缀表达式的结果 值。



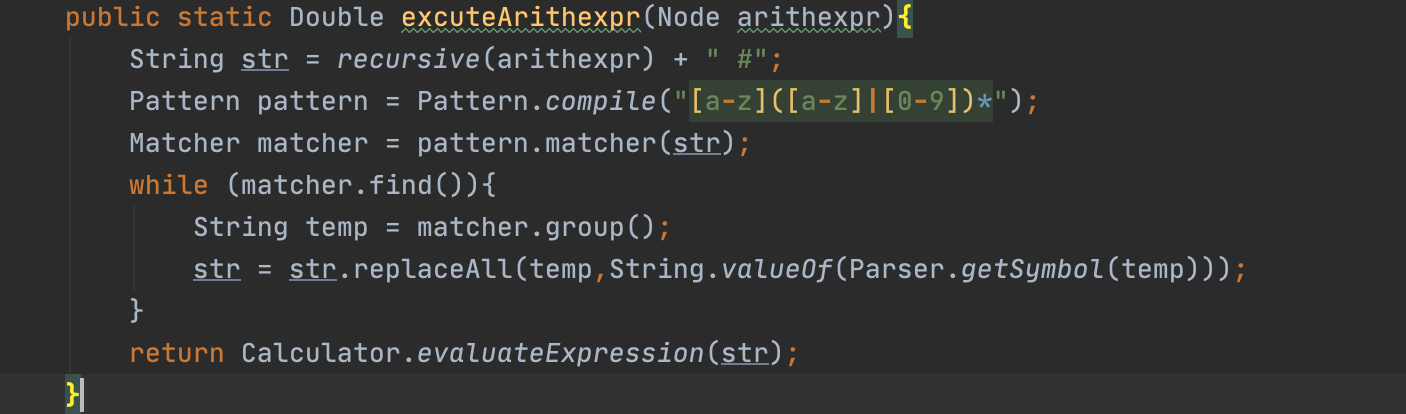
在Node节点里，我定义了一系列计算非终结符节点的方法。如下图所示。



这里列出几个比较特殊的节点的处理。

1. excuteArithexpr

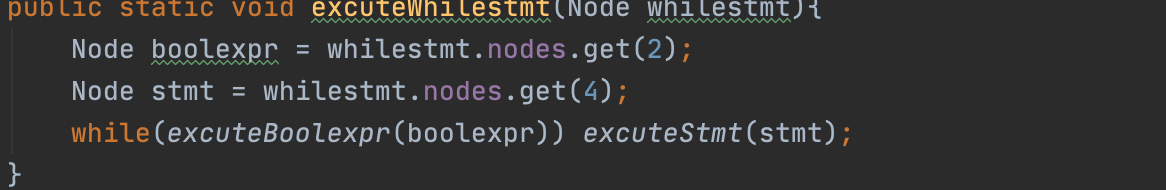
先通过先序遍历将arithexpr节点表示的字符串计算出来。然后再将表达式字 符串中的变量用符号表中的数字替换掉，接着调用Caculator中的 excuteExpression来计算出表达式的结果值，然后返回结果。



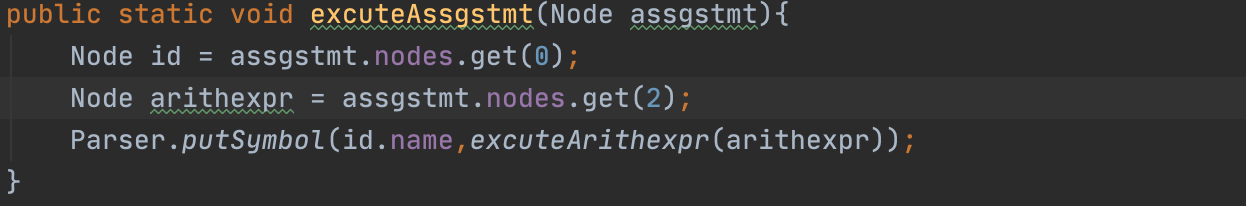
2.excuteIfstmt



3.excuteWhilestmt



4.excuteAssgstmt

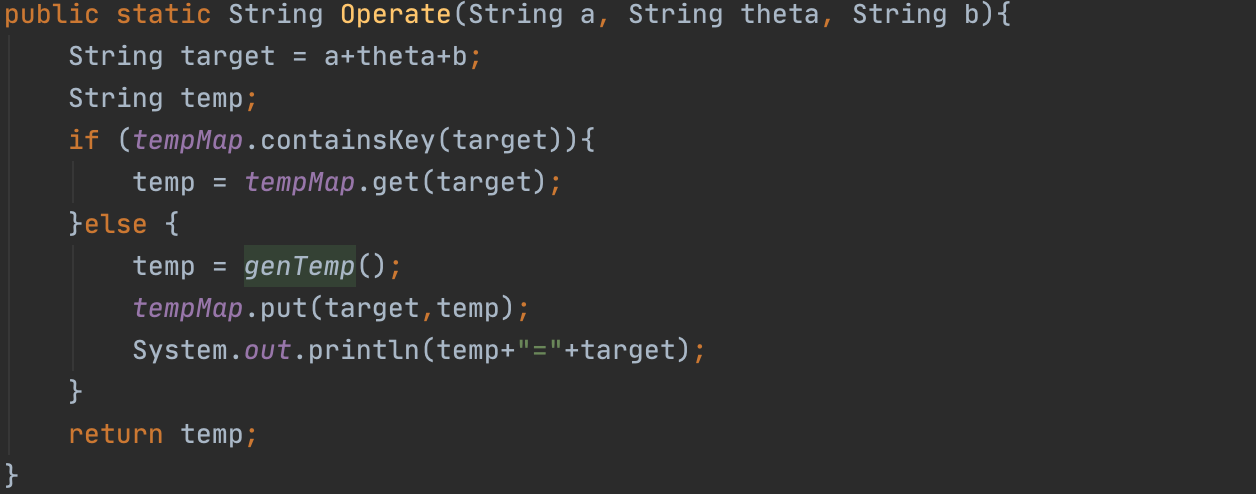


最后，只需要在调用makeProgram函数，传入抽象语法树的根节点，就可以递归调用excute-xx函数进行语义分析了。

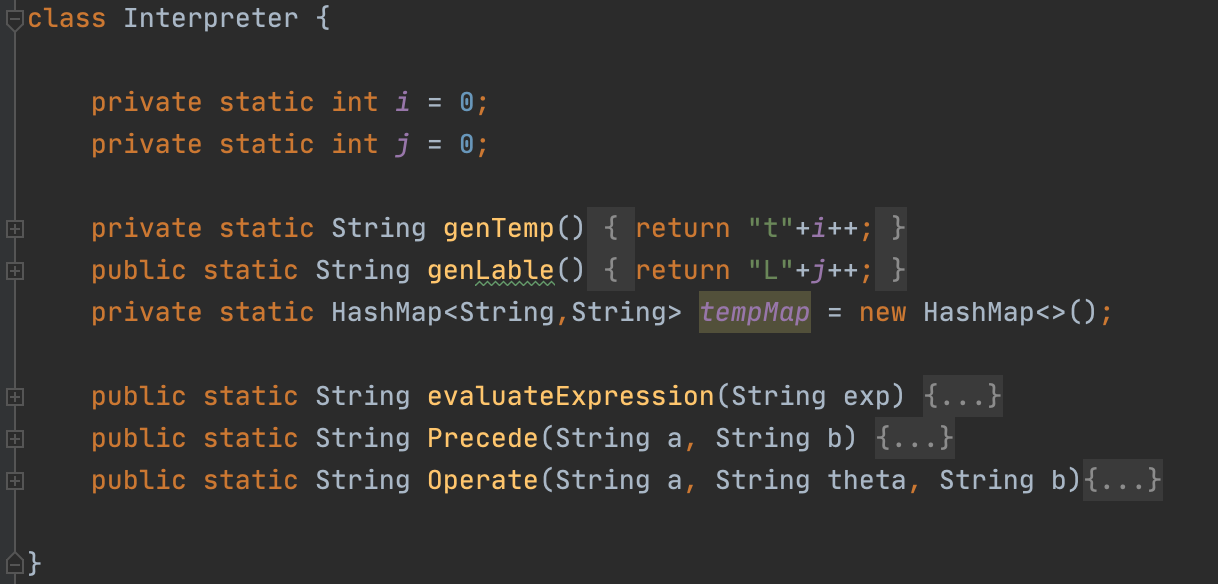
* 1. 代码生成：
     1. 代码生成方法；

中间代码采用的是三地址码

还是利用的将arithexpr中的表达式通过中缀转后缀的方式进行生成三地址代 码，利用后缀表达式计算的特点，每当栈顶放入一个运算符的时候，就将最上 面的两个数字弹出然后计算结果，这个结果放在了ti里，并且在一个表达式中 对相同的基运算进行了存储优化(在生成ti变量之前，先去tempMap里寻找， 有没有相同的基运算表达式，如果有则不用新生成，直接用原来的就好了)



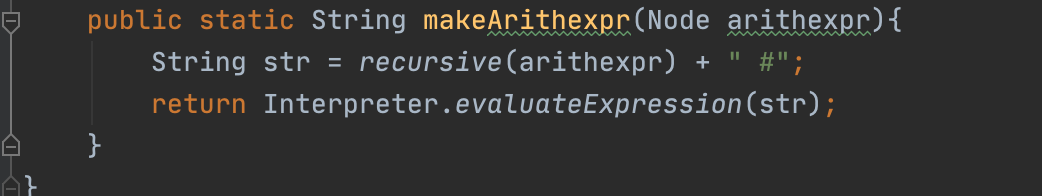
每当完成对一个arithexpr的三地址代码生成之后，都需要将tempMap清空。 对中间变量t和中间标识段L的生成变量都是递增的。



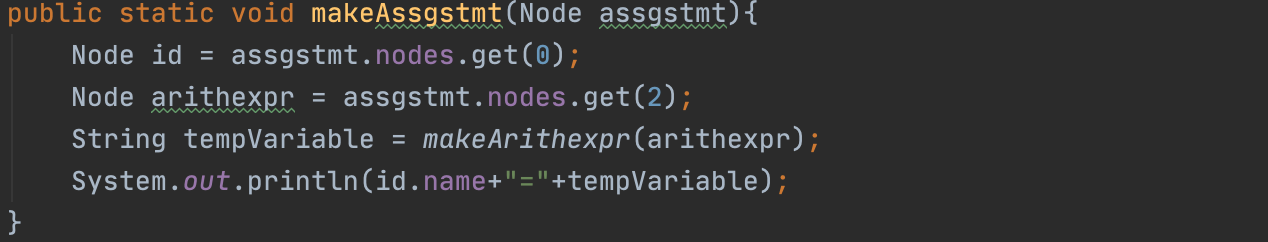
整个三段地址生成的过程都是递归执行的，还是利用语法生成树来递归生成， 同语义分析比较相似，不过这次调用的不是excutexx方法，而是makexx方 法。

这里列出几个比较特殊的节点的三段地址代码生成的方法

1. makeArithexpr



1. makeAssgsmt



3.makeIfstmt



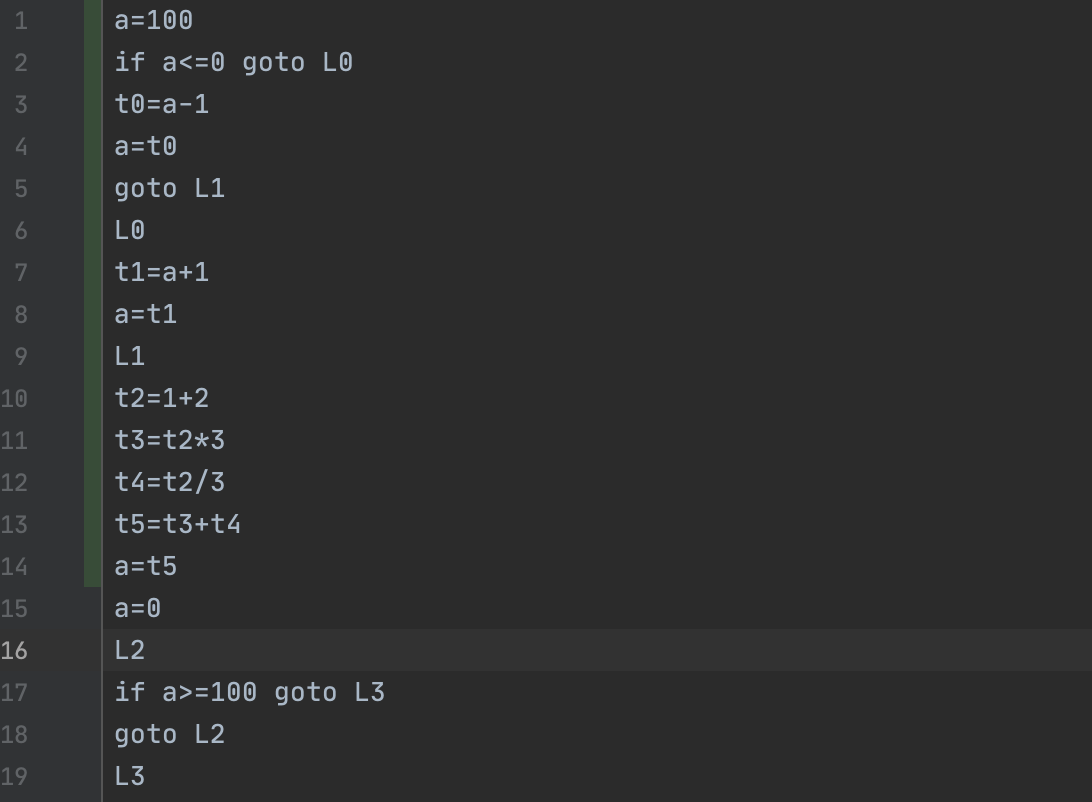
4.makeWhilestmt



最后只需要调用makeProgram并将语法生成树的根节点传入就能递归生成三段地址代码了。

* + 1. 代码输出

生成的三段地址代码如下图所示:



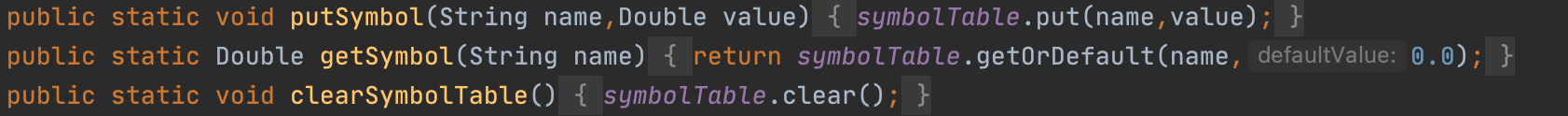
* 1. 符号表：
     1. 符号表的结构；

我用的hashMap这样的数据结构来生成的符号表，其键值对为String和 Double

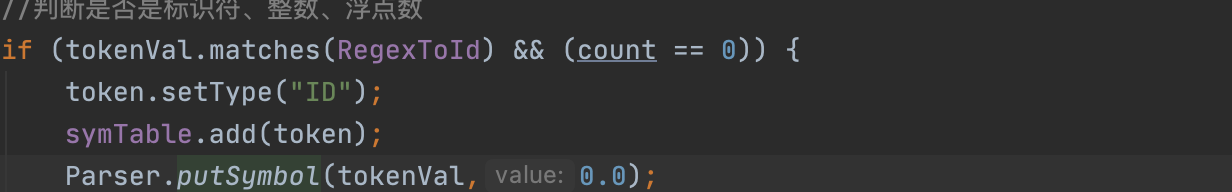


符号表的查询、添加、修改等操作

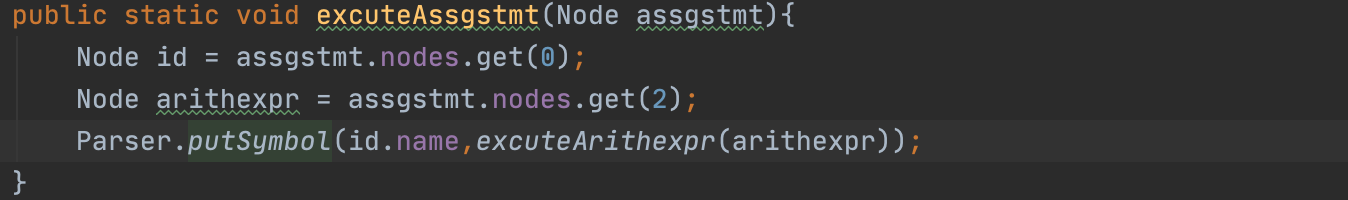
如图所示，定义了符号表的三种操作方法，分别是添加，获取和清空



在进行词法分析的时候，如果读取到了一个类型为ID的token，就会将其加 入符号表中，并设置其初始值为0.0



在excuteAssgstmt方法中，当对一个ID进行赋值的时候，便会在符号表里 进行添加或修改。



在计算arithexpr节点的值的时候，如果表达式里含有ID变量，需要先通过调 用getsymbol来将变量替换为符号表中的值，然后再计算其中缀表达式的结 果。

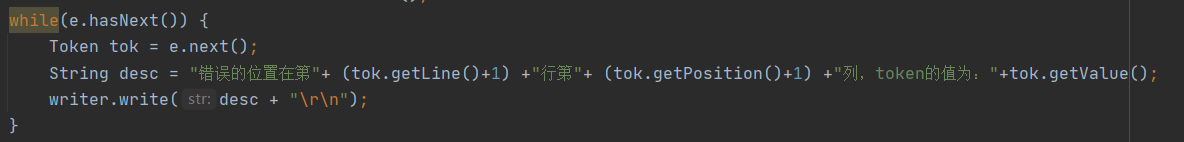
* 1. 出错处理：
     1. 出错信息；

出错信息会通过错误处理函数输出到文件，并有错误对应的位置和错误提示信息



* + 1. 出错定位

通过token的position和line进行出错定位



1. 功能实现（输入/输出显示、界面效果等）
   1. 基本的输入、输出界面；

输入：可以手动输入，也能上传文件，在线读取文件内容；还能直接清空，同时可以进行主题的切换





输出：点击运行后会展示tokenlist，语法生成树，中间代码，符号表，同时还有错误提示（Build Message）



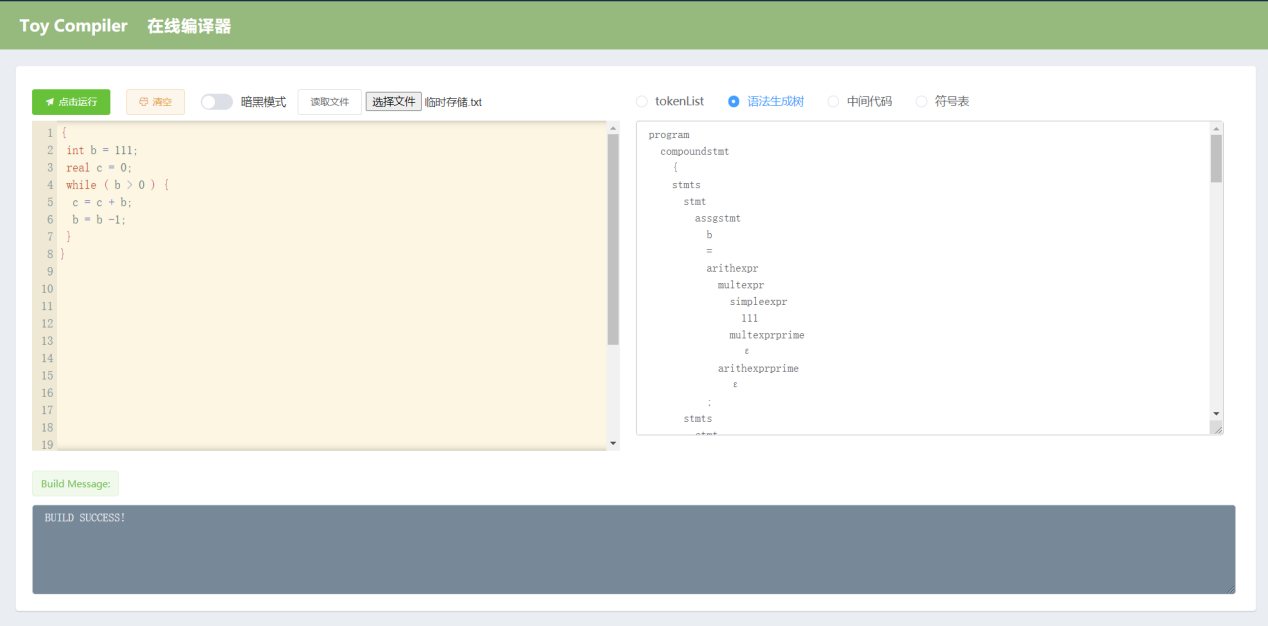


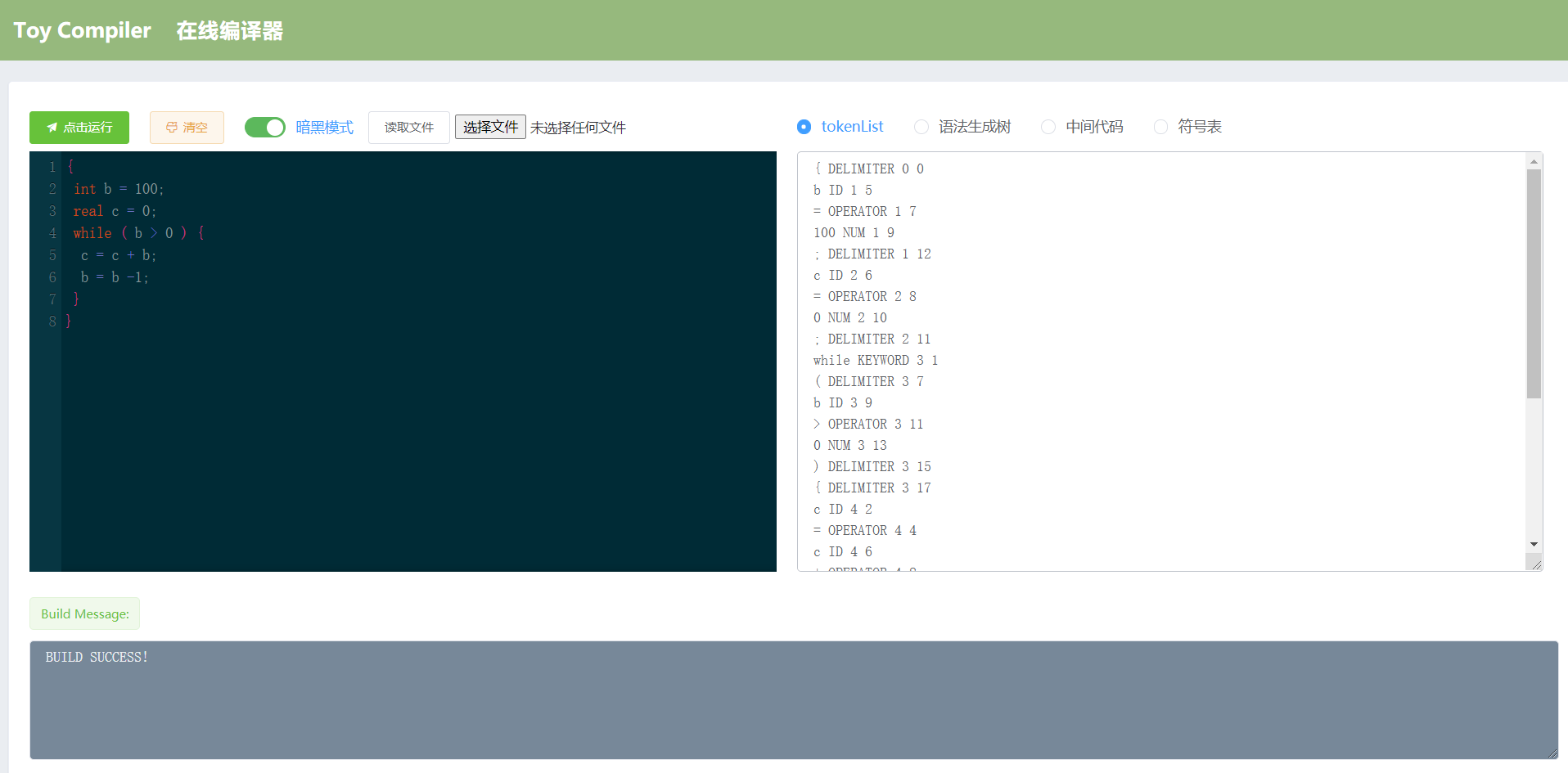






* 1. 界面布局与设计（自由发挥）





左边为输入界面，右边为输出界面，底部是输出信息提示，采用flex进行布局，同时代码输入界面采用了codemirror插件，实现代码高亮，自动补全，主题切换功能

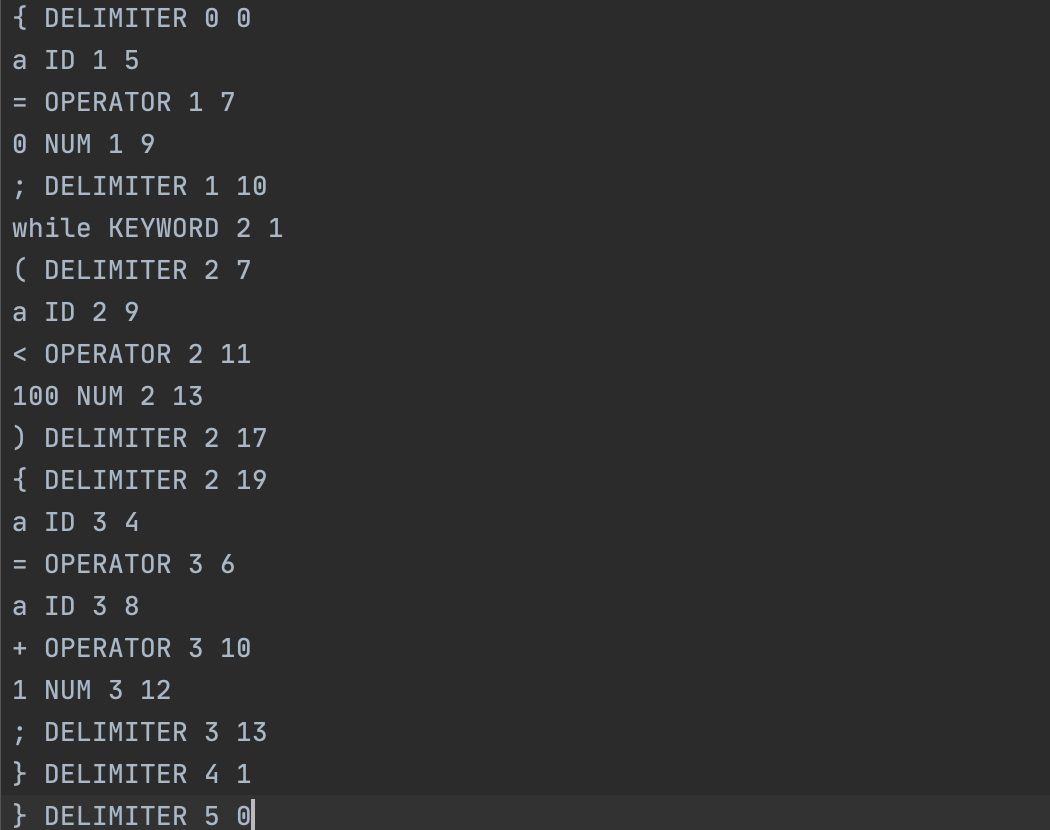
1. 测试（测试用例等）
   1. 自测用例

测试样例一:(循环，赋值)

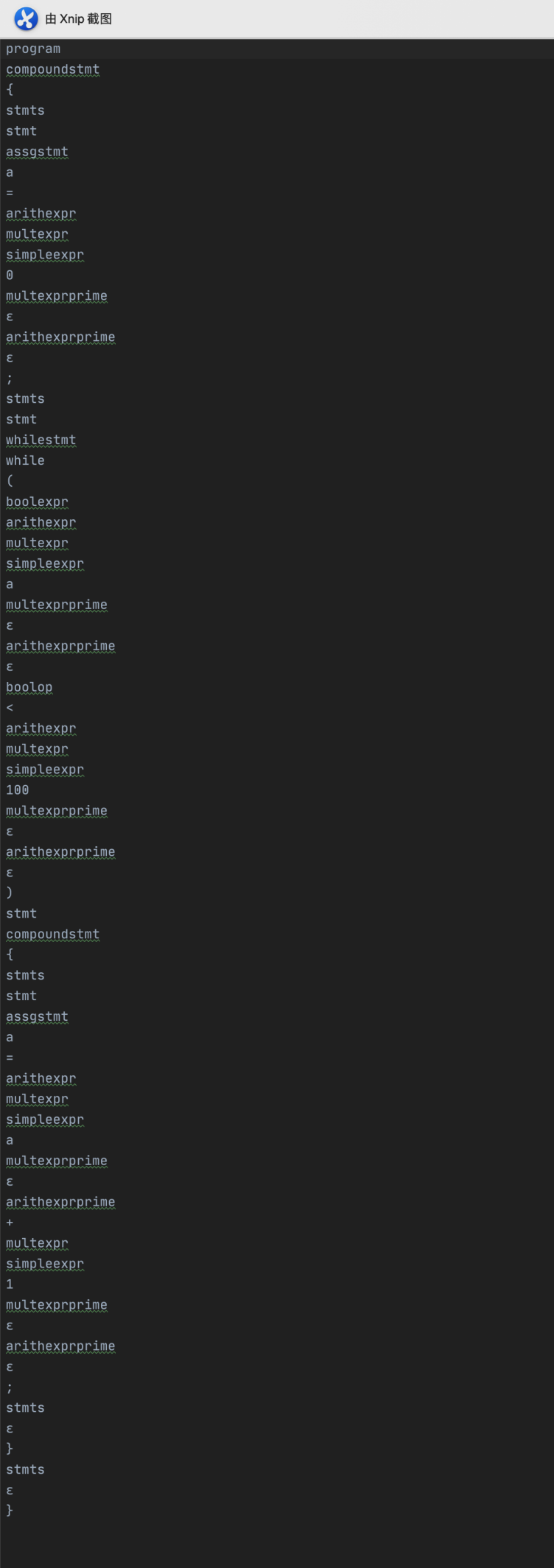
1.代码段



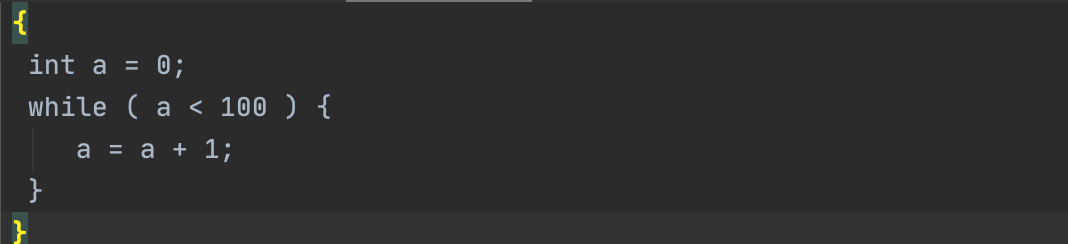
1. tokenlist



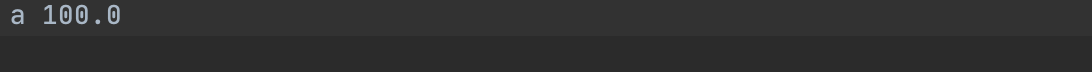
3. parseTree



4. inner code

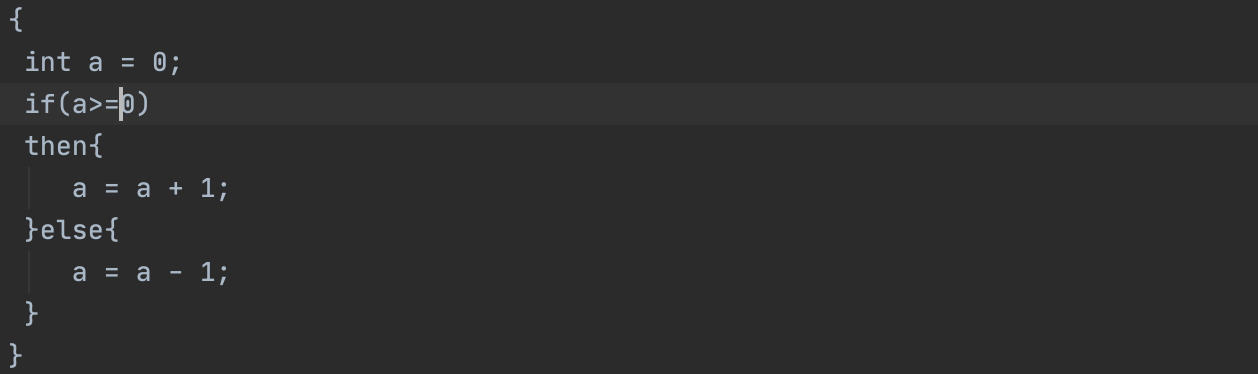


5. symbol table

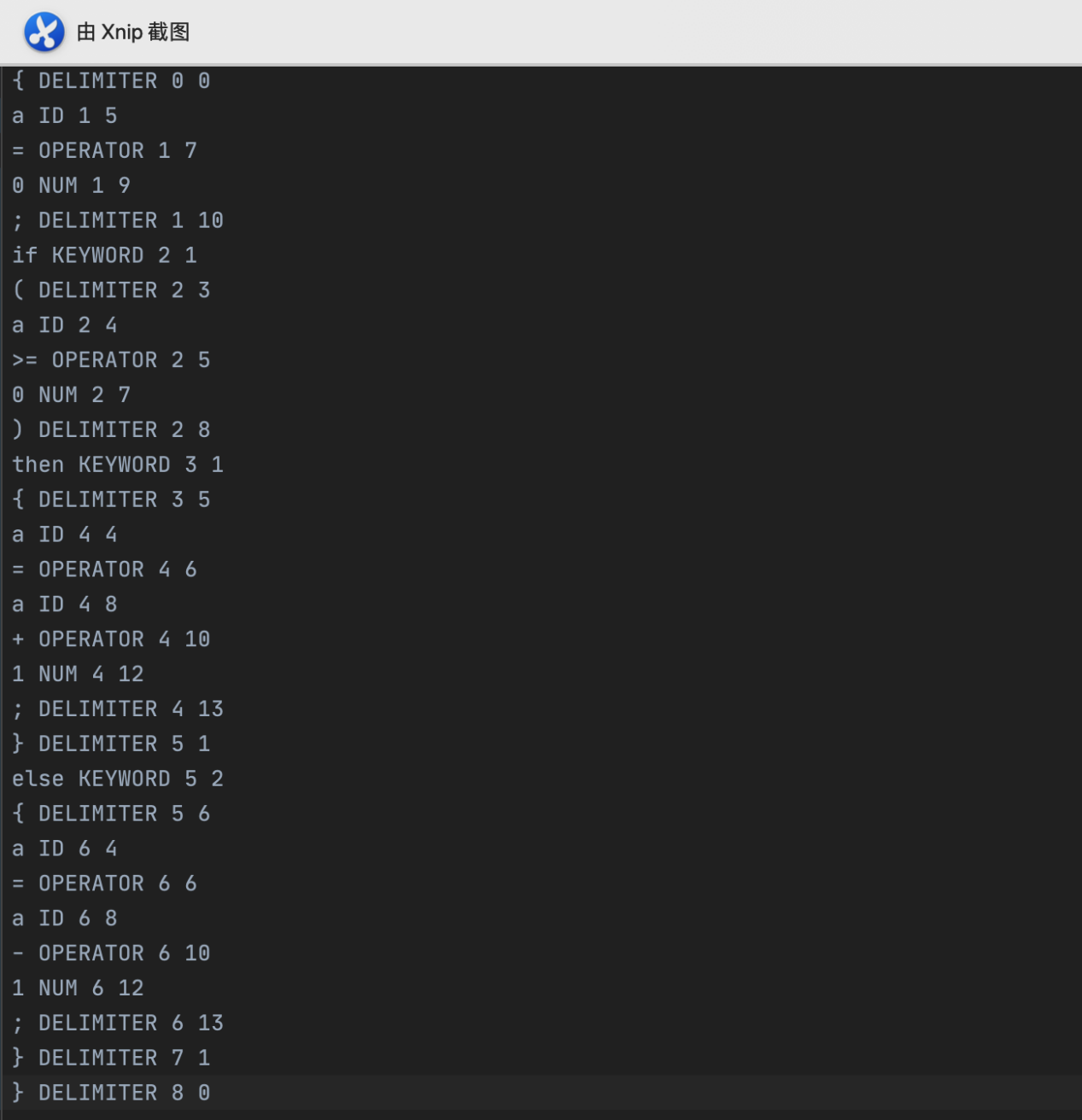


测试样例一:(判断，赋值)

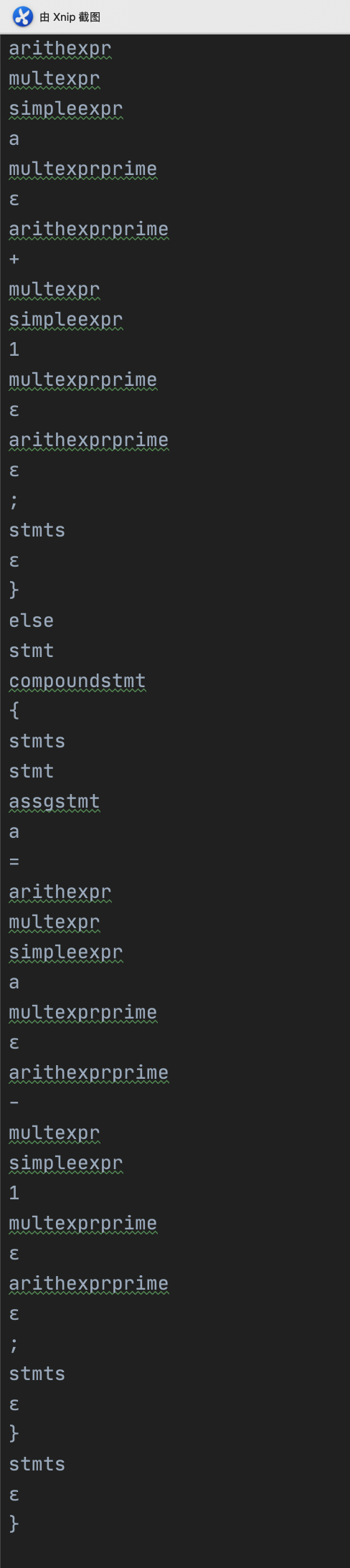
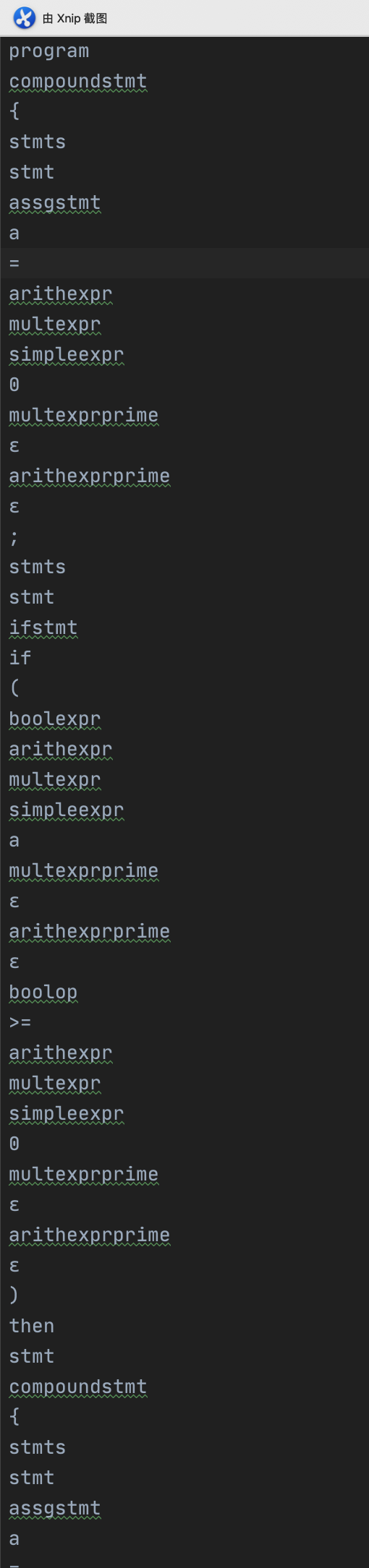
1.代码段



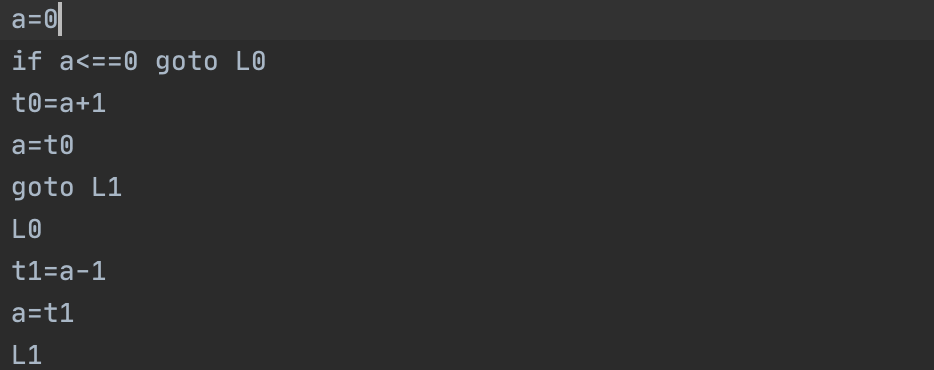
2.tokenlist



3. parseTree



4. inner code



5. symbol table



* 1. 检查用例

通过读入文件内容功能读入检查用例文件（LLtest2018.txt），点击运行进行查看



运行结果：

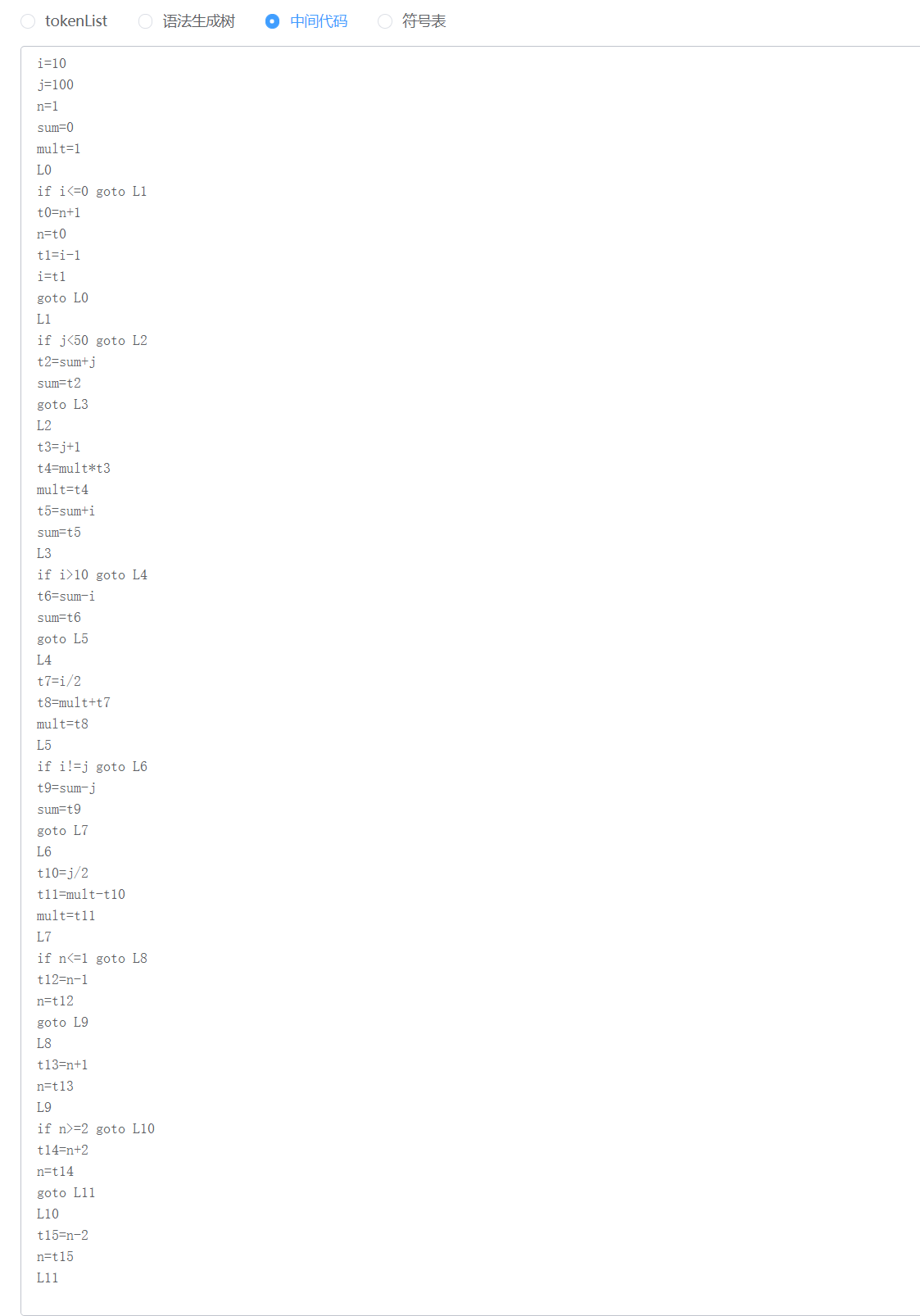
tokenList：



语法生成树：



中间代码：



符号表：



1. 总结（总结技术、结果等）

技术总结:词法部分通过利用子集构造法(正则表达式->NFA->DFA)，最小化状态图完成了对toy语言中token的识别，然后再利用书上的LL(1)语法技术(构造项目集，语法分析表，first集合，fellow集合等)结合语法分析栈完成语法检验。在语法推导翻译时采用了属性文法和前缀表达式转后缀表达式的技术，并通过递归语法生成树的方式完成了语法推导翻译。并且按照书上所介绍的DAG三段地址生成代码的方式完成了中间代码的生成。最后用Java完成了真个toy编译器的编写。

刚开始做实验的时候比较生疏，感觉无从下手，后面通过看书和查阅资料对实验有了大概的了解，然后开始尝试和队友分任务，写步骤、流程和程序，渐入佳境。通过这次实验让我对词法分析的理解更深了一步。通过本次课程设计，增加了我们的动手能力和程序分析能力，同时在此过程中，体会到的一个项目的进行，一个软件的开发的大体流程以及基本任务过程。在此过程中，加深了对DFA的理解与运用，状态机之间的转换，理解了正则表达式的关键。最重要的是增加了对代码实现以及项目开发的兴趣。

不足:

能检测的程序比较简单，而且报错功能也是有限，不能拿来实际运用。程序设计中，存在很多冗余，不够简洁，而且注释较少，增加了程序的阅读难度。

1. 参考文献

1.Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman-Compilers - Principles, Techniques, and Tools-Pearson\_Addison Wesley (2006)

1. 工作量占比

黄俊豪： 50%

彭潇： 50%