

软件课程设计（II）

实验报告

学 院：计算机科学与工程学院

班 号： 9181069501

姓 名： 孙同

学 号： 918106840543

指导老师： 王永利

2021年5月

目录

[1. 实验简介 3](#_Toc70948138)

[1.1 实验环境 3](#_Toc70948139)

[1.2 文件目录 4](#_Toc70948140)

[1.3 程序简介 4](#_Toc70948141)

[2. 词法分析 5](#_Toc70948142)

[2.1 简述 5](#_Toc70948143)

[2.2 词法文件 5](#_Toc70948144)

[2.3 数据结构 7](#_Toc70948145)

[2.4 总流程图 8](#_Toc70948146)

[2.5 技术细节 9](#_Toc70948147)

[2.5.1 NFA合并 9](#_Toc70948148)

[2.5.2 NFA转DFA 9](#_Toc70948149)

[3 语法分析 10](#_Toc70948150)

[3.1 简述 10](#_Toc70948151)

[3.2 语法文件 10](#_Toc70948152)

[3.3 数据结构 11](#_Toc70948153)

[3.4 总流程图 11](#_Toc70948154)

[3.4.1 构建LR1项目集族 11](#_Toc70948155)

[3.4.2 语法分析 12](#_Toc70948156)

[3.5 技术细节 13](#_Toc70948157)

[3.5.1 计算first集 13](#_Toc70948158)

[3.5.2 计算闭包 13](#_Toc70948159)

[4 语义分析 14](#_Toc70948160)

[4.1 简述 14](#_Toc70948161)

[4.2 数据结构 14](#_Toc70948162)

[4.3 总流程图 16](#_Toc70948163)

[4.4 技术细节 16](#_Toc70948164)

[4.4.1 抽象语法树生成 16](#_Toc70948165)

[4.4.2 符号表生成 16](#_Toc70948166)

[4.4.3 控制语句——拉链与代码回填技术 17](#_Toc70948167)

[5 运行结果 17](#_Toc70948168)

[5.1 词法运行结果 17](#_Toc70948169)

[5.1.1 正确识别科学计数、复数、浮点数等 17](#_Toc70948170)

[5.1.2 异常处理 17](#_Toc70948171)

[5.2 语法分析结果 18](#_Toc70948172)

[5.2.1 语法正确 18](#_Toc70948173)

[5.2.2 语法错误 18](#_Toc70948174)

[5.3 语义分析结果 19](#_Toc70948175)

[5.3.1 语义正确 19](#_Toc70948176)

[5.3.2 异常处理 20](#_Toc70948177)

[5.4 嵌套符号正确引用结果 21](#_Toc70948178)

[6 感想心得 22](#_Toc70948179)

# 实验简介

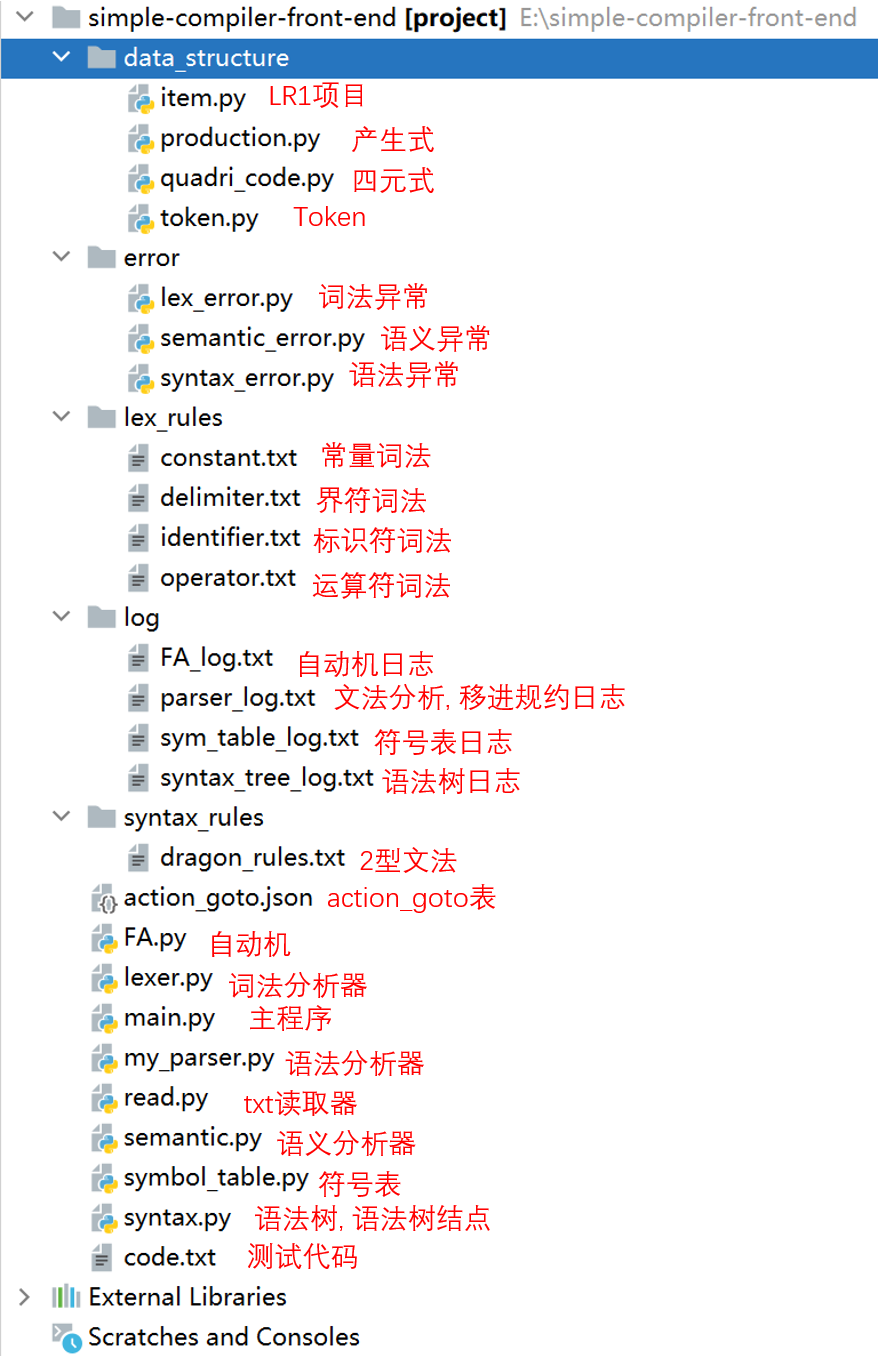
编译程序（或编译器、编译系统）在计算机科学与技术的发展历史中发挥了巨大作用，是计算机系统的核心支撑软件。在上学期的《编译原理》课程中，我们学习了程序设计语言编译程序构造的一般原理、基本设计方法和主要实现技术。而在本学期尝试去实现一个简单的编译程序前端：包括词法分析器、语法分析器和语义分析器。

## 实验环境

**语言**：Python 3.8

**IDE**：PyCharm

## 文件目录



## 程序简介

本次实验使用Python设计了一个**词法分析器、语法分析器、语义分析器**。

**词法分析器（lexer）**：读取lex\_rules下各个Token类型的正规词法，分别构造NFA（不确定有限自动机）。然后拼接各个NFA，将其转换为DFA（确定有限自动机）。随后读取测试代码，逐个字符扫描并将其送入DFA，最终输出一个Token序列（行号，类别，Token内容）。

**语法分析器（parser）**：读取syntax\_rules下的dragon\_rules.txt，首先生成其增广文法，然后求取各个非终结符的first集，构建LR1项目集族，利用项目集族生成action-goto表并将其保存为json文件便于多次使用，最后利用action-goto表对词法分析器输出的Token序列进行分析。

**语义分析器（semantic analyzer）**：语义分析写在了程序中，但容易对其进行修改。语义分析器首先在之前语法分析的过程中生成AST（抽象语法树），随后对语法树进行深度优先搜索，逐结点生成符号表，进行继承属性、综合属性的计算，最终输出一个四元式序列。

**异常处理（exception）**：继承Python的Exception类。在词法、语法、语义分析的过程中，如果出现了一些错误便会抛出这些自定义异常，终止后续分析。

程序大量使用**面向对象的思想**，是为了方便优化和使用。

代码已随附件上传，同时也可在Github上获取<https://github.com/stfromnjust/simple-compiler-front-end>

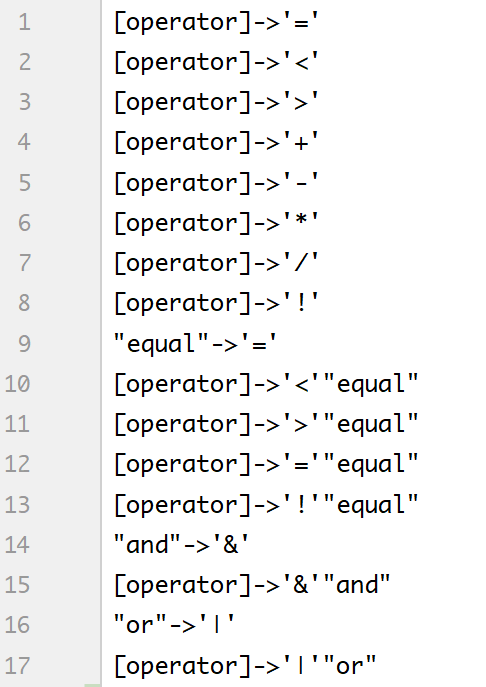
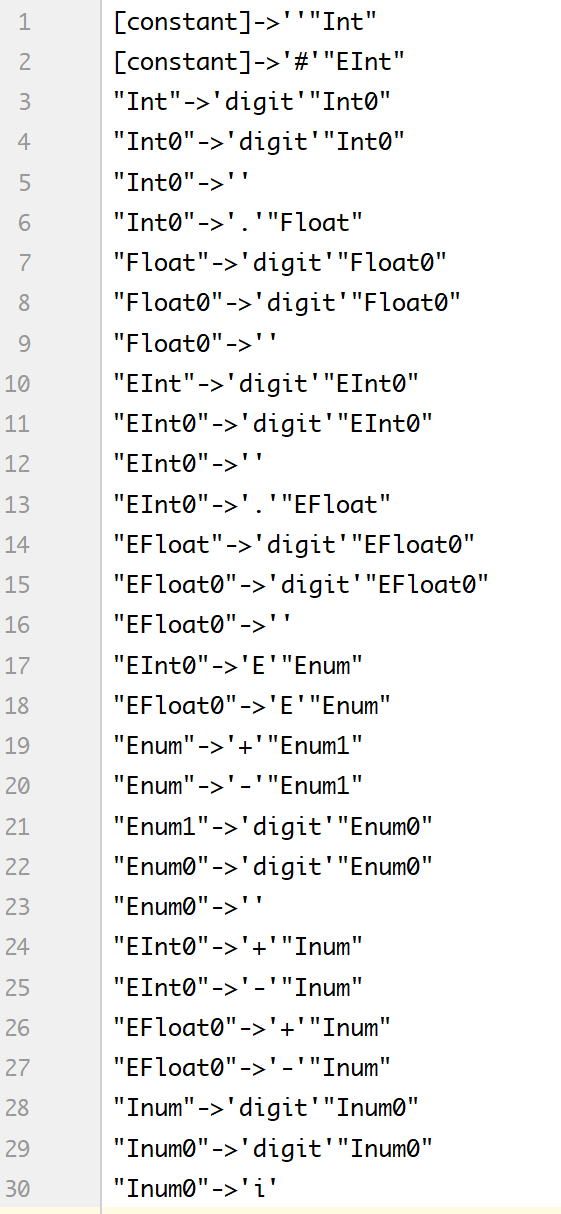
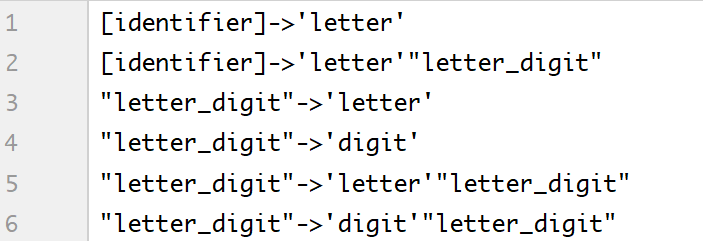
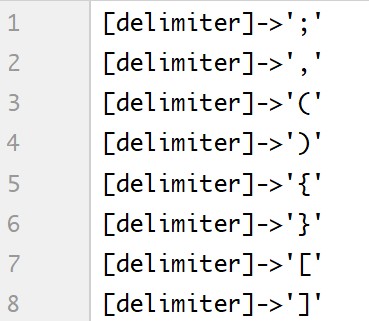
# 词法分析

## 简述

**词法分析器（lexer）**根据用户输入的正规文法生成NFA，再确定化生成DFA，根据DFA识别Token，从头到尾从左至右识别用户输入的源代码，生成Token列表（三元组：所在行号，类别，Token内容）

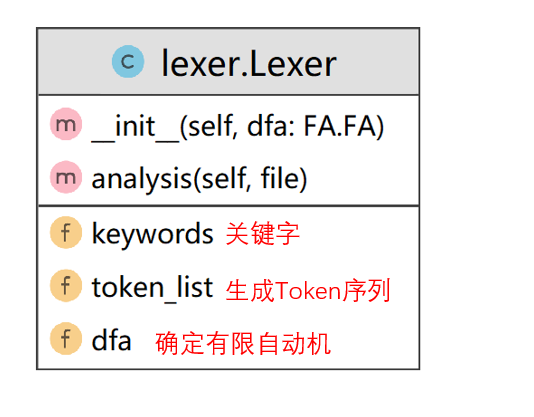
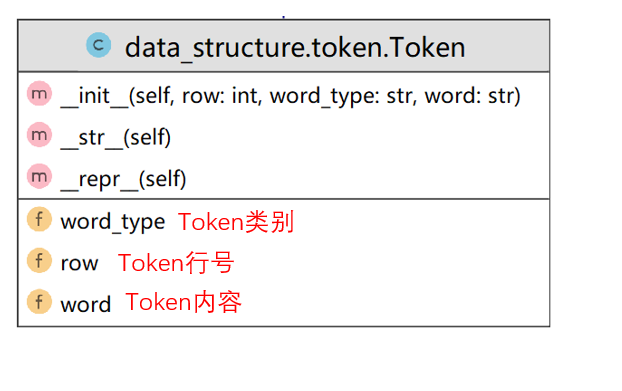
词法分析器涉及到以下文件：data\_structure/token.py、error/lex\_error.py、lex\_rules/constant.txt，delimiter.txt，identifier.txt，operator.txt、FA.py、lexer.py、read.py、main.py、code.txt。

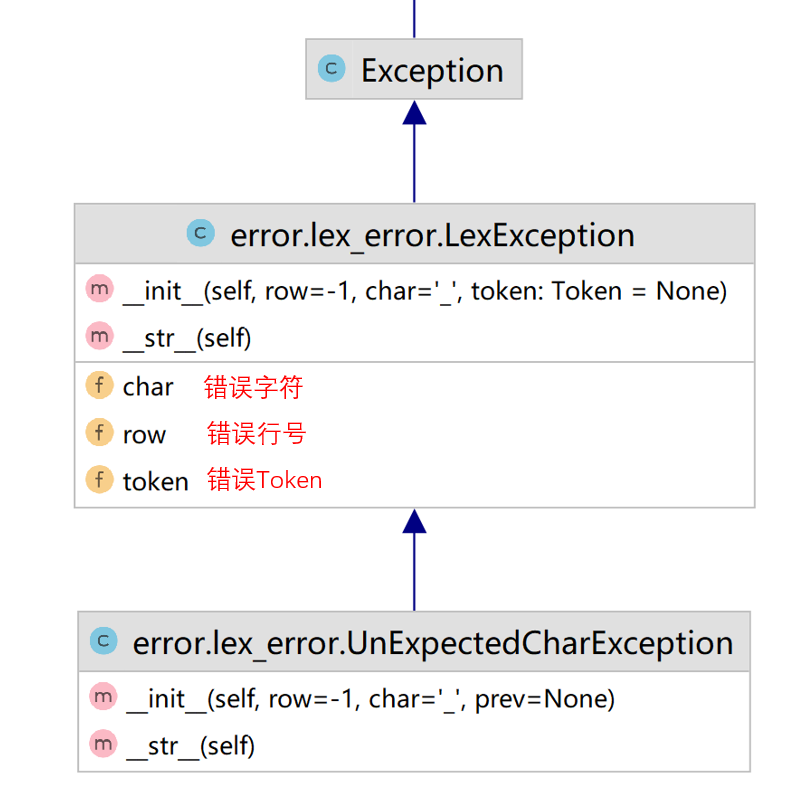
## 词法文件



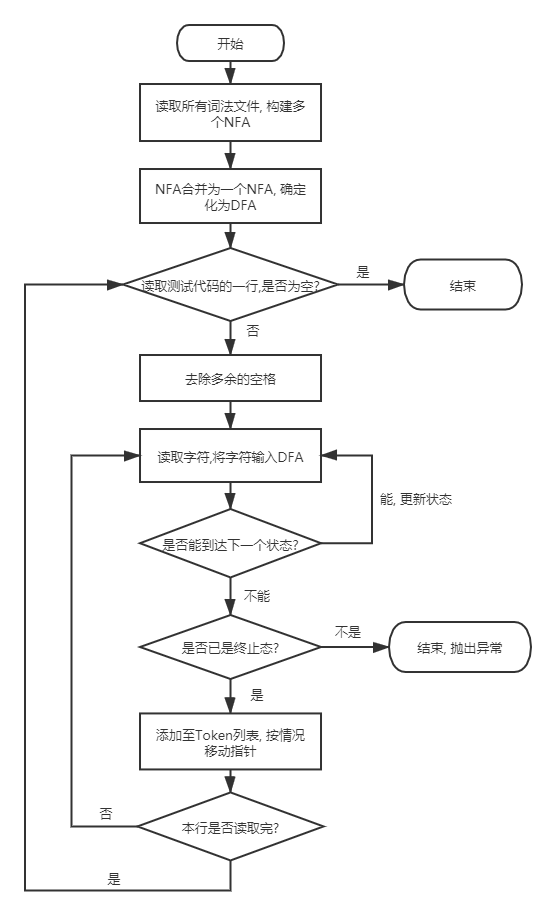
## 2.3 数据结构







## 总流程图

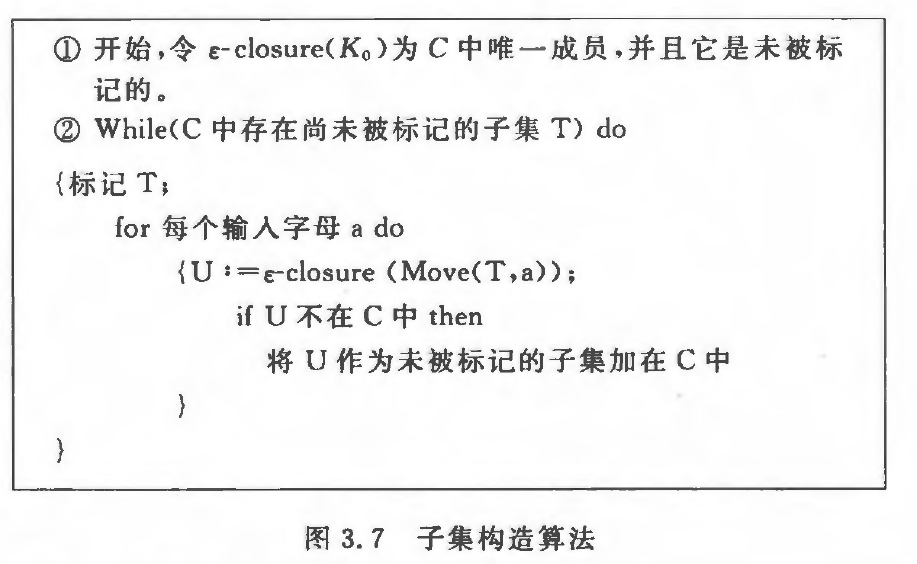


## 技术细节

### NFA合并



### NFA转DFA



# 语法分析

## 3.1 简述

**语法分析器（syntax parser）**是编译过程的第二个阶段。语法分析的任务是在词法分析的基础上将单词序列分解成各类语法短语，如“程序”、“语句”、“表达式”等。

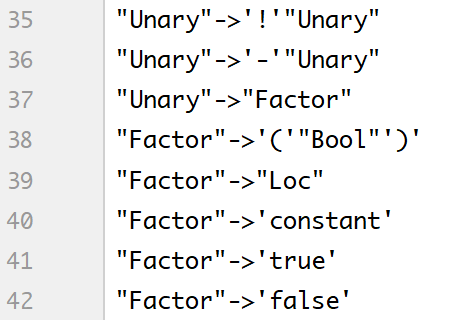
语法分析有多种方法，主要分为自顶向下的分析方式和自底向上的分析方式。其中自底向上的分析方式又分为LR（0）、SLR（1）、LR（1）、LALR（1）分析等。

本次实验采用LR（1）分析

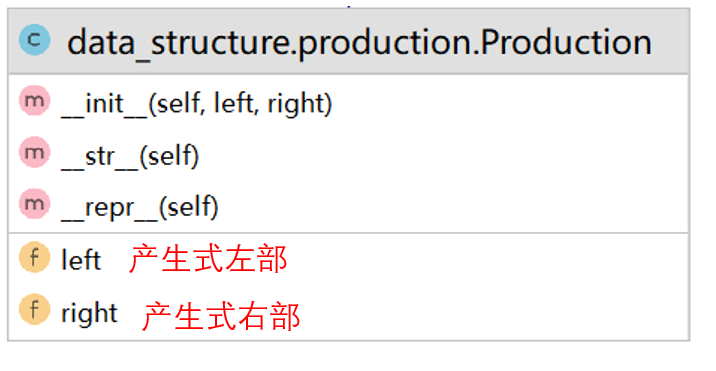
LR1语法分析器涉及到以下文件：data\_structure/production.py, item.py, token,py、error/syntax\_error.py、syntax\_rules/dragon\_rules.txt、action\_goto.json、main.py、my\_parser.py、read.py、code.txt

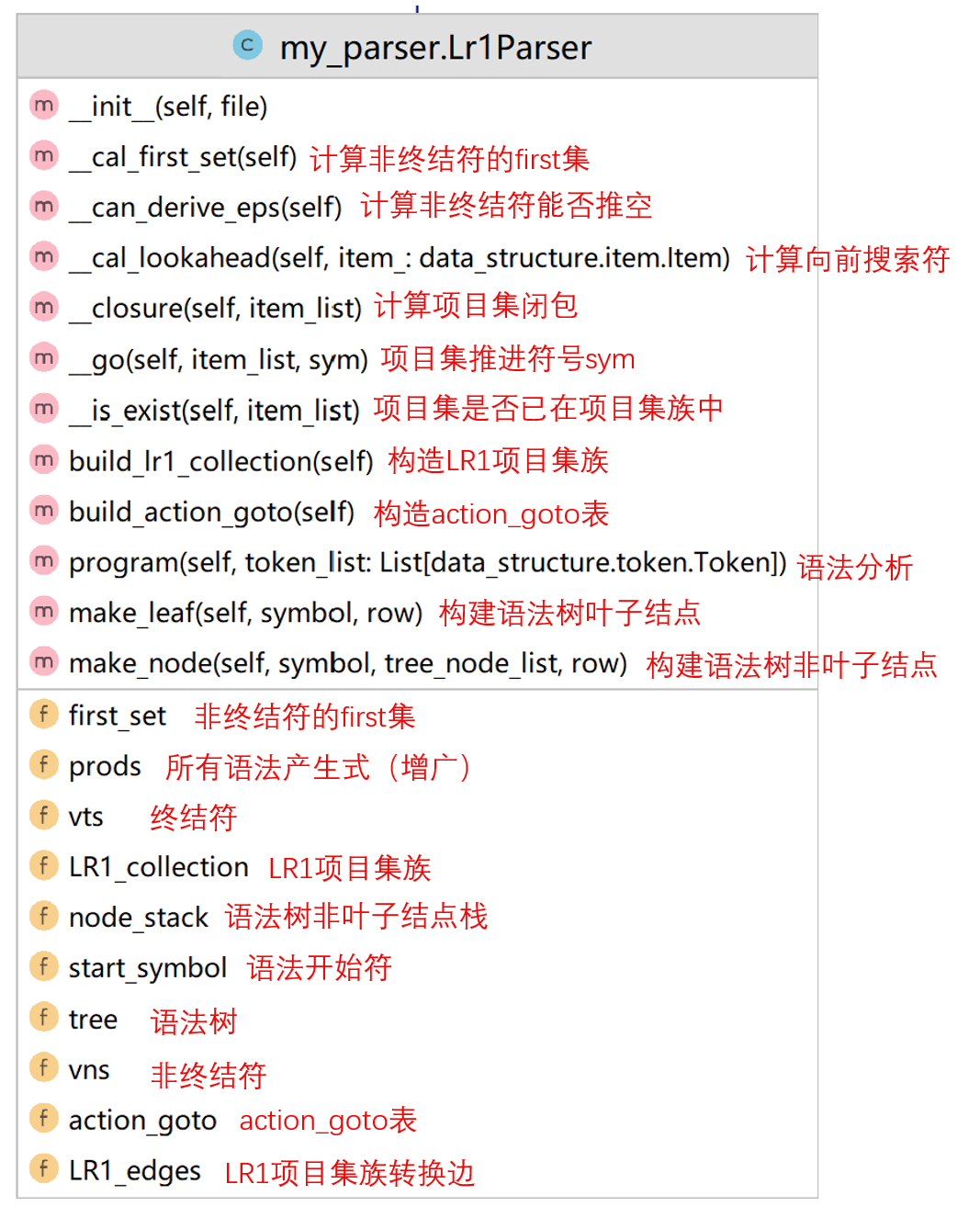
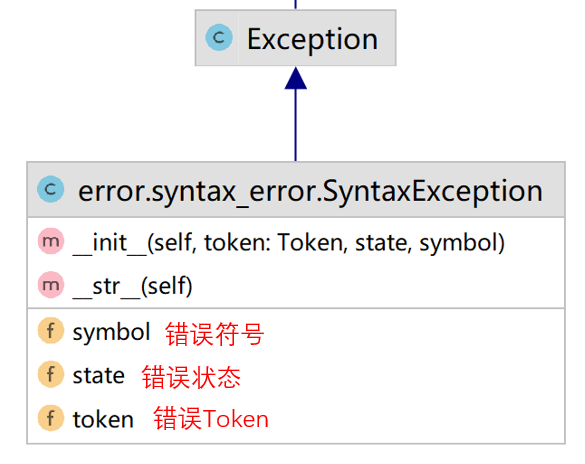
## 3.2 语法文件

此语法文件参考了《编译原理第2版》（“龙书”）附录一中的语法。因而命名为dragon\_rules。



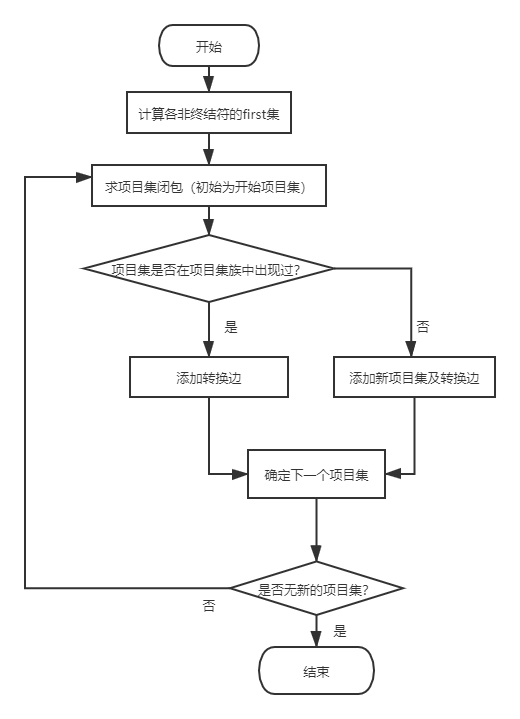
## 3.3 数据结构





## 3.4 总流程图

### 3.4.1 构建LR1项目集族



### 3.4.2 语法分析

## 技术细节

### 计算first集

**Step 1：**求出能推出空的非终结符。设非终结符能推出空的数组为X

1. 将X中的每一非终结符的标记设初值为“未定”
2. 扫描文法中的产生式
   1. 删除所有右部含有终结符的产生式，若这使得某一非终结符为左部的所有产生式都被删除，将对应左部非终结符的标志置为“否”。
   2. 若某一非终结符某一产生式右部为空，将数组中对应该非终结符的标志置为“是”，并从文法中删除该非终结符所有的产生式。
3. 扫描产生式右部的每一符号
   1. 若扫描到的非终结符在数组中对应为“是”，则删去该非终结符，若这使产生式右部为空，则将产生式左部的非终结符的标志置为“是”，并删除以该非终结符为左部的所有产生式。
   2. 若扫描到的非终结符在数组中对应为“否”，则删去该产生式。若这使得某一非终结符为左部的所有产生式都被删除，将对应左部非终结符的标志置为“否”。
4. 重复3，直到扫描完一遍文法的产生式，数组无变化且没有值为“未定”的非终结符。

**Step 2：**计算first集

1 若XVn，且有产生式X->a...，aVt，则afirst(X)

2 若XVn，X->，则X

3 若X，Y1，Y2，…YnVn，而有产生式X->Y1Y2…Yn。当Y1，Y2，…Yi->时，first（Y1）- {}，first（Y2）- {}..., first（Yi-1）- {}，first（Yi）都包含在first（X）中。

4 若3中所有Yi->, 则first（X）= first（Y1）∪ first（Y2）∪…∪first（Yn）∪ {}

5 重复上述步骤，直至每个first集都不再增大为止。

### 3.5.2 计算闭包

1. 遍历项目集中的所有项目

2. 如果某项目的点后为非终结符，则添加所有以该非终结符为左部，点位置在第0位的项目，同时按书上法则计算向前搜索符

3. 不断添加，直到项目集中的项目不再增加为止

# 语义分析

## 4.1 简述

语义分析（semantic analyzer）是审查源程序有无语义错误，为代码生成阶段收集类型等信息，并利用这些信息生成中间代码的一个过程。

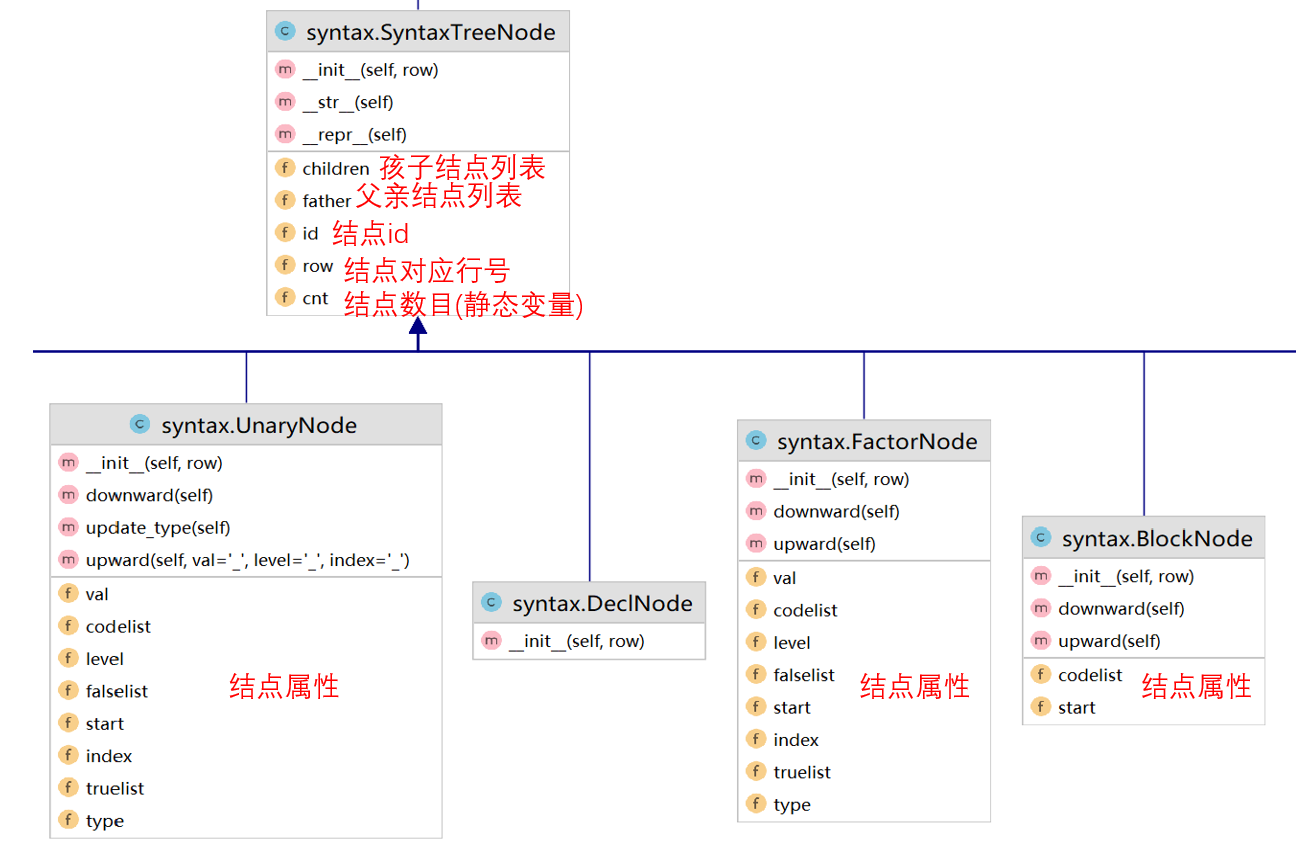
本程序中虽然将语义分析写在了代码中（而非txt），但是有着很强的拓展性，方便扩展属性文法。

本次实验中，语义分析器的实现分为如下几个步骤：首先在语法分析的过程中顺便生成抽象语法树（AST），然后对抽象语法树进行深度优先搜索，在搜索过程中会实时更新符号表，树中各个结点的继承属性和综合属性，最后会输出最终的四元式序列。

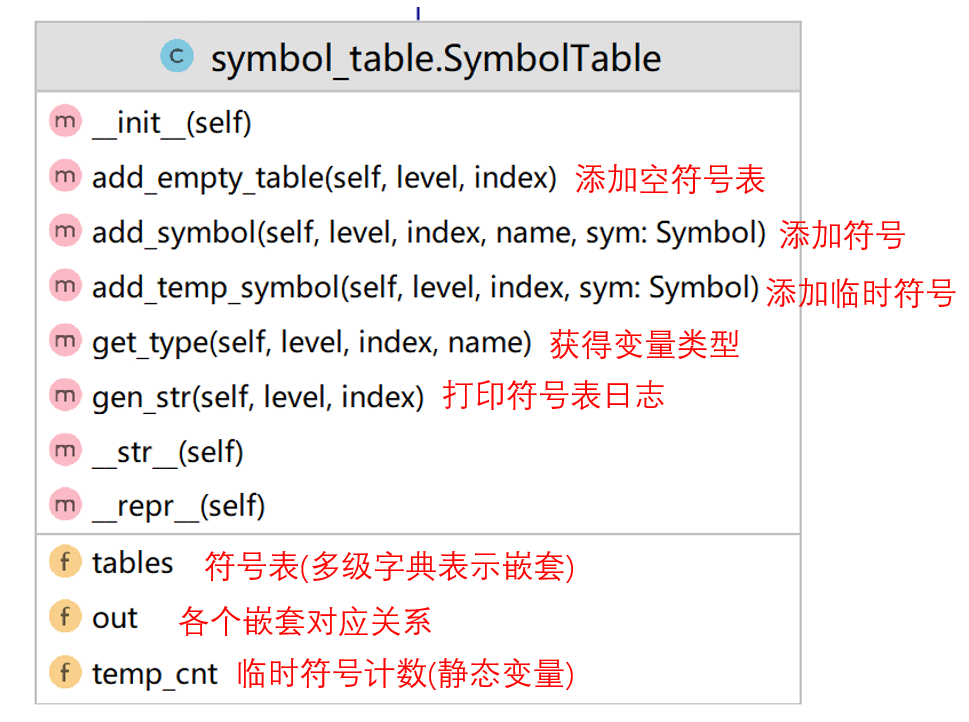
本次实验支持算数表达式的正确计算，if语句，if-else语句，while语句的正确控制流程，赋值语句的正确生成。

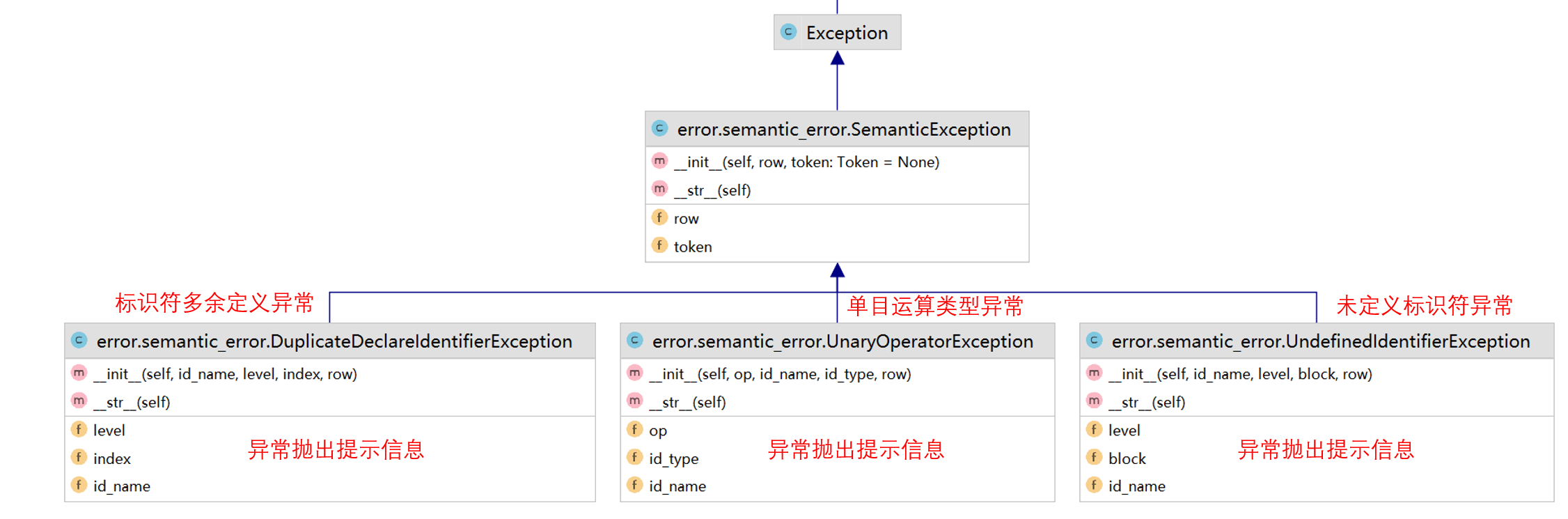
语义分析器涉及到以下文件: data\_structure/quadri\_code.py、main.py、semantic.py、symbol\_table.py、syntax.py、error\semantic\_error.py。

## 4.2 数据结构



完整SyntaxTreeNode的uml：<https://ftp.bmp.ovh/imgs/2021/05/433804f09c99c1a4.png>



## 4.3 总流程图

## 4.4 技术细节

### 4.4.1 抽象语法树生成

每当进行规约的时候，扫描规约产生式右部的符号：如果符号为终结符则生成一个叶子结点添加到列表X中，如果符号为非终结符则弹出结点栈（node\_stack）中的一个结点添加到列表X中。最后生成一个结点，其孩子结点列表为X，并将其压入结点栈。

按照此算法可以生成正确的语法树。

### 4.4.2 符号表生成

由于文法特性是每个Block中都会有声明语句Decl，因此当遇到Block结点时需要记录当前Block嵌套层次，更新该嵌套层次的序号。遇到Decl结点时需要在符号表当前嵌套层次，序号的位置添加符号信息。

### 4.4.3 控制语句——拉链与代码回填技术

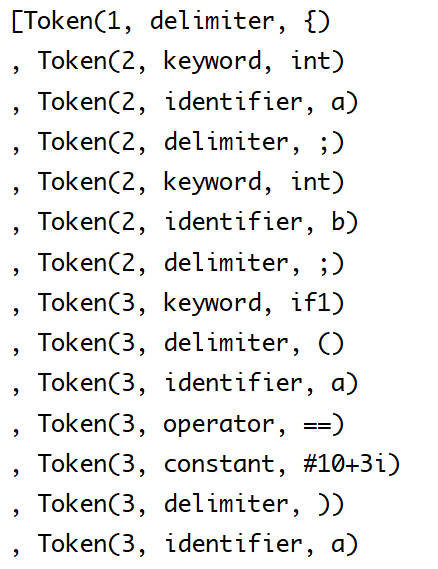
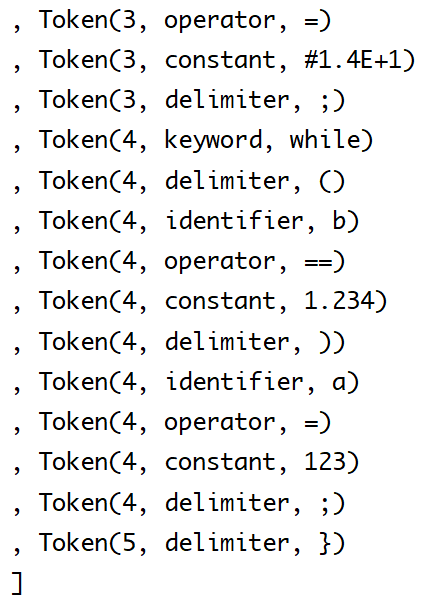
当处理到某一步，生成的转移语句不能确定目标语句标号时，先将目标语句标号的位置用‘\_’表示，并将该转移语句的地址加入到某个链表（truelist，falselist，nextlist）中；当这个目标语句标号可以确定之时，再将其回填至‘\_’处。

# 运行结果

## 5.1 词法运行结果

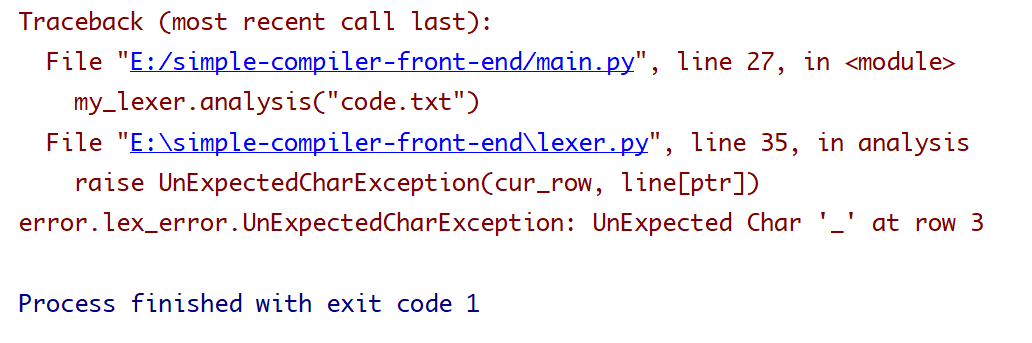
### 5.1.1 正确识别科学计数、复数、浮点数等

1. {
2. int a; int b;
3. if1 (a == #10+3i)a=#1.4E+1;
4. while (b == 1.234) a=123;
5. }

### 5.1.2 异常处理

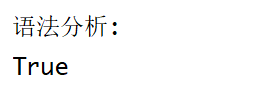
1. {
2. int a; int b;
3. a = \_a;
4. }



## 5.2 语法分析结果

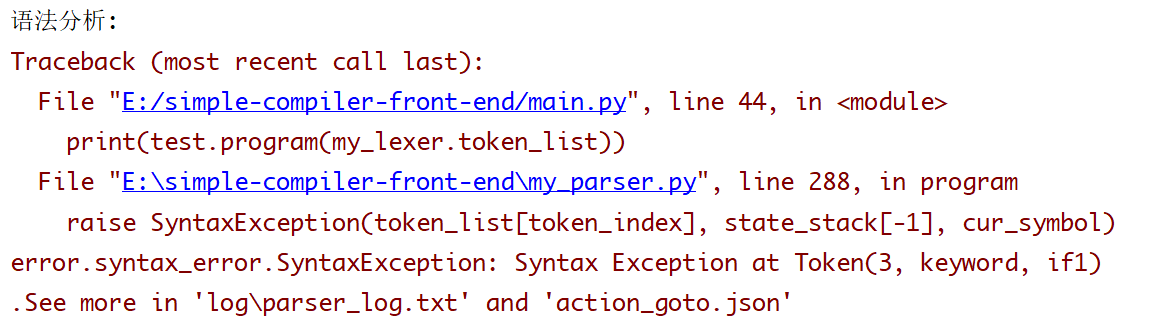
### 5.2.1 语法正确

1. {
2. int a; int b;
3. if (a==1)
4. {
5. a = a + 1;
6. }
7. while (b>0 && a < 3)
8. {
9. b = b \* (-2) / 3;
10. }
11. }



### 5.2.2 语法错误

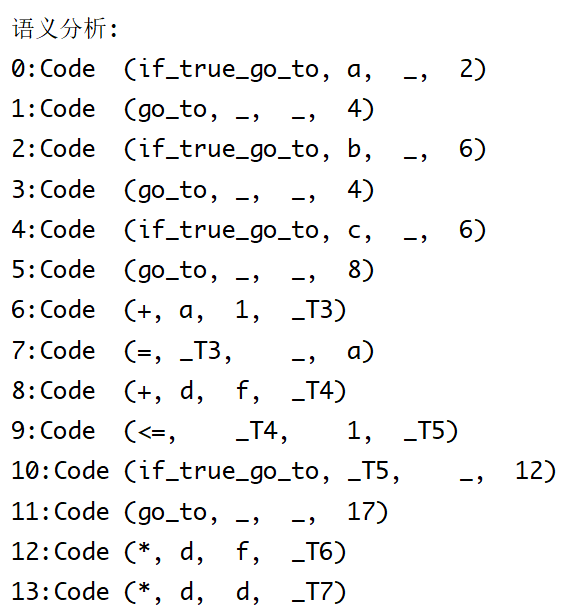
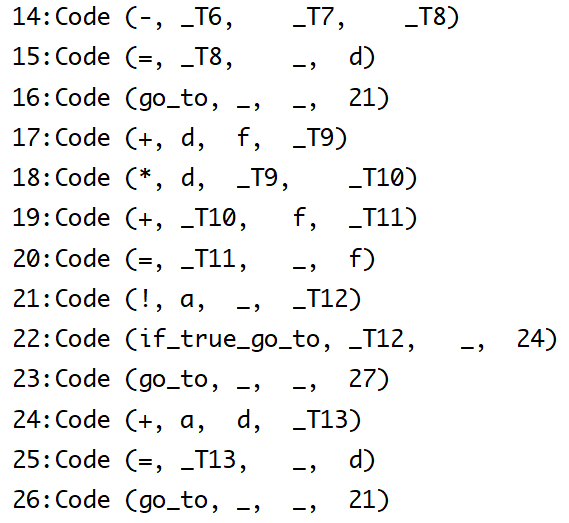
1. {
2. int a; int b;
3. a = a + if1;
4. }



## 5.3 语义分析结果

### 5.3.1 语义正确

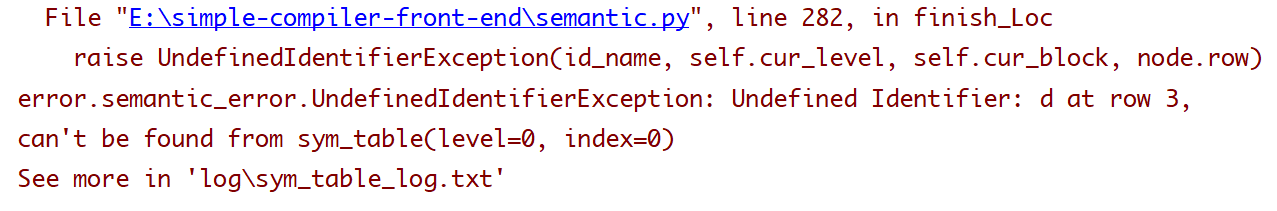
1. {
2. bool a; bool b; bool c;
3. int d; int f;
4. if1 (a && b || c)
5. {
6. a = a + 1;
7. }
8. if2 (d+f<=1)
9. {
10. d = d \* f - d \* d;
11. }
12. else
13. {
14. f = d \* (d + f) + f;
15. }
16. while (!a)
17. {
18. d = a + d;
19. }
20. }

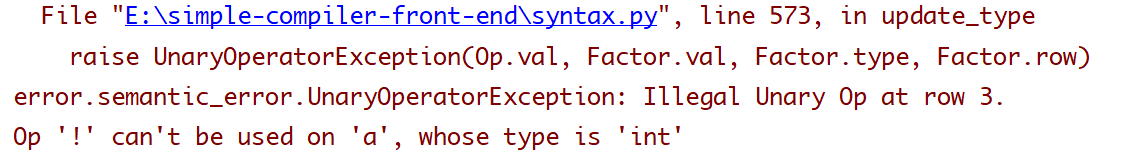


### 5.3.2 异常处理

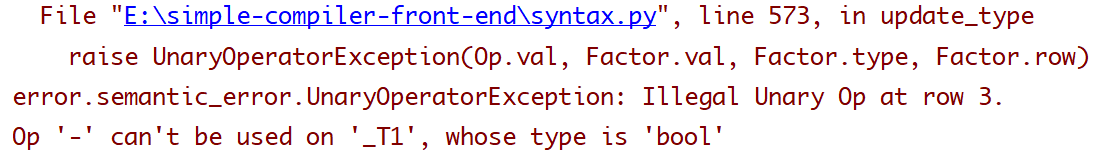
**1. 未定义标识符异常**

1. {
2. int a; int b;
3. d = 1;
4. }

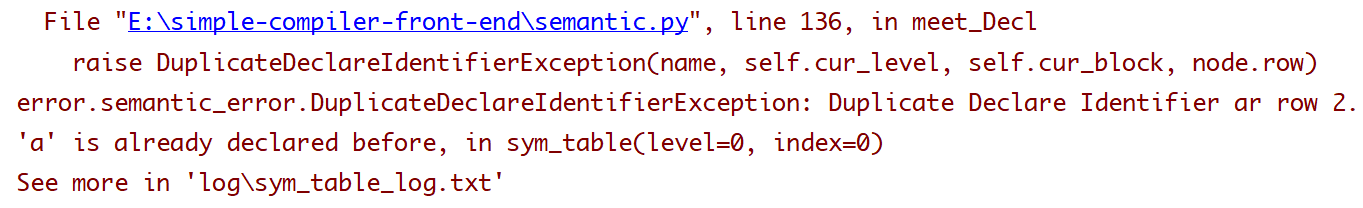


1. **单目运算符类型异常**
2. {
3. int a; int b;
4. a = !a;
5. }

1. {
2. bool a; bool b;
3. a = -(a&&b);
4. }

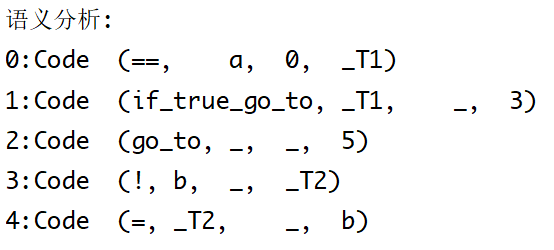


1. **重复定义标识符异常**
2. {
3. bool a; int b; int a; int b;
4. }



## 5.4 嵌套符号正确引用结果

1. {
2. int a; int b;
3. if1 (a == 0)
4. {
5. bool b;
6. b = !b;
7. }
8. }



# 感想心得

**心路历程**：

这次软件课设总共耗费的时间加起来差不多用了9天。（词法分析2天+语法分析3天+语义分析3天+写报告整理1天）。

这9天是一个从茫然无措到成就感十足的一个过程。除了如何读取txt参考了一下网上的方法，其余的包括各种分析器的设计以及数据结构的构建均由本人独立完成。虽说代码质量可能欠佳，但好歹也算是实现了一个完整的编译器前端。

**成长进步**：

通过这次软件课设，首先，我认为我的python代码编写水平得到了很大的提高，同时对代码进行调试和Debug的速度也比以前快了不少。

然后是代码的架构能力，巩固并加强了面向对象的设计思想，让程序看起来比以前更整洁、有条理。但仍需继续努力，希望能写出质量更高的代码。

最后就是耐心与毅力，在编写语法分析器和语义分析器的时候我遇到过很多Bug以及数据结构设计上的问题在中途需要对代码进行大的改动。曾也一度想要放弃“抵抗”，不过还好最后还是坚持下来了。

**Bug回顾**：

这次软件课设让我印象最深的Bug，就是设计LR1项目的数据结构时，我一开始打算让Item类继承Production类。但是当用Set去装这些Item时会陷入死循环。后来通过查询资料得知，python的set是通过Hashset实现的，因此set中的数据结构必须是hashable的，也就是说Item类中不能有python列表，而且需要正确的重写类的\_\_hash\_\_( )方法。代码也因此不得不进行大改动。

处于好奇我又查询了其他语言的类似情况，发现C++中的Set提供有红黑树的实现方法，我也因此体会到了不同程序语言之间的微妙差异。这个python知识点也被我牢记在心中，怕是永远也不会忘了。

**遗憾**：

这次软件课设的遗憾第一是由于时间因素，我的词法、语法、语义都是比较简单的，没有进行更深一步的钻研和打磨。第二则是来不及做四元式序列的优化了，很可惜。