

**[C++&fLex]**

**编译原理语法分析**

**实验报告**

**学院：计算机学院**

**2020211376 马天成**

**2022年11月18日**

**北京邮电大学《编译原理》课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验**  **名称** | 语法分析 | | **学 院** | 计算机 | **指导教师** | 刘辰 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2020211305** | **2** | **2020211376** | | **马天成** |  | |
| **实**  **验**  **内**  **容** |  | | | | | |
| **学生**  **实验**  **报告** | （详见“实验报告和源程序”册） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

目录

[1. 实验目的和要求 4](#_Toc119706395)

[1.1 实验目的 4](#_Toc119706396)

[1.2 实验要求 4](#_Toc119706397)

[2. LL1程序 4](#_Toc119706398)

[2.1 在词法分析程序上扩展程序 4](#_Toc119706399)

[2.2 模块设计 4](#_Toc119706400)

[2.3 Main函数 5](#_Toc119706401)

[2.4 Lex-识别进入各类词法函数 5](#_Toc119706402)

[2.5 Grammar类详解 6](#_Toc119706403)

[2.5.1 Grammar-获得文法输入信息 7](#_Toc119706404)

[2.5.2 Grammar-消除左公因子和左递归 8](#_Toc119706405)

[2.5.3 Grammar-寻找First和Follow 8](#_Toc119706406)

[2.5.4 Grammar-构造文法分析表 9](#_Toc119706407)

[2.6 Parser类详解 10](#_Toc119706408)

[2.6.1 Parser-读入词法分析结果到栈 10](#_Toc119706409)

[2.6.2 Parser-分析过程 10](#_Toc119706410)

[2.6.3 Parser-输出 10](#_Toc119706411)

[2.7 运行结果 11](#_Toc119706412)

[3. YACC 11](#_Toc119706413)

[3.1 Lex部分 11](#_Toc119706414)

[3.2 YACC部分 12](#_Toc119706415)

[3.3 运行结果 13](#_Toc119706416)

[4. 问题&解决 13](#_Toc119706417)

[4.1 Lex->Parser 13](#_Toc119706418)

[4.2 First和Follow递归 13](#_Toc119706419)

[4.3 栈的分析过程 14](#_Toc119706420)

[5. 实验总结 14](#_Toc119706421)

# 1. 实验目的和要求

## 1.1 实验目的

语法分析程序的设计与实现

## 1.2 实验要求

编写语法分析程序，实现对算术表达式的语法分析。要求所分析的算术表达式 由如下的文法产生： 在本实验中，我们需要用LL(1)和LR两种方法来对输入串进行分析 对于LL(1)方法，要求如下：

（1）为给定文法自动构造预测分析表

（2）构造LL(1)预测分析程序 对LR方法，要求如下：

（1）构造识别该文法所有活前缀的DFA

（2）构造该文法的LR分析表

（3）构造LR分析程序

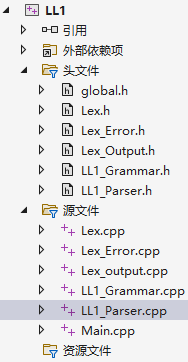
构造YACC程序自动生成文法分析程序

# 2. LL1程序

## 2.1 在词法分析程序上扩展程序

首先是词法分析。词法分析的结果是把输入文件变为符号和符号类型以及坐标信息的元组。所以我们直接在main函数里进行词法分析后，再进行语法分析。

词法分析这里就不再过多赘述。

 但词法分析结果一开始是打算读取输出文件的，但是输出文件是系统调用，后面执行的时候还没有对文件进行修改，所以我们只能用全局变量来解决。

## 2.2 模块设计

lex, output, error有对应.h&.c

1. global.h：规定数据类型

2. Main.c：主函数，运行

3. Lex：词法分析模块

4. Lex\_Output：输出正确词法模块

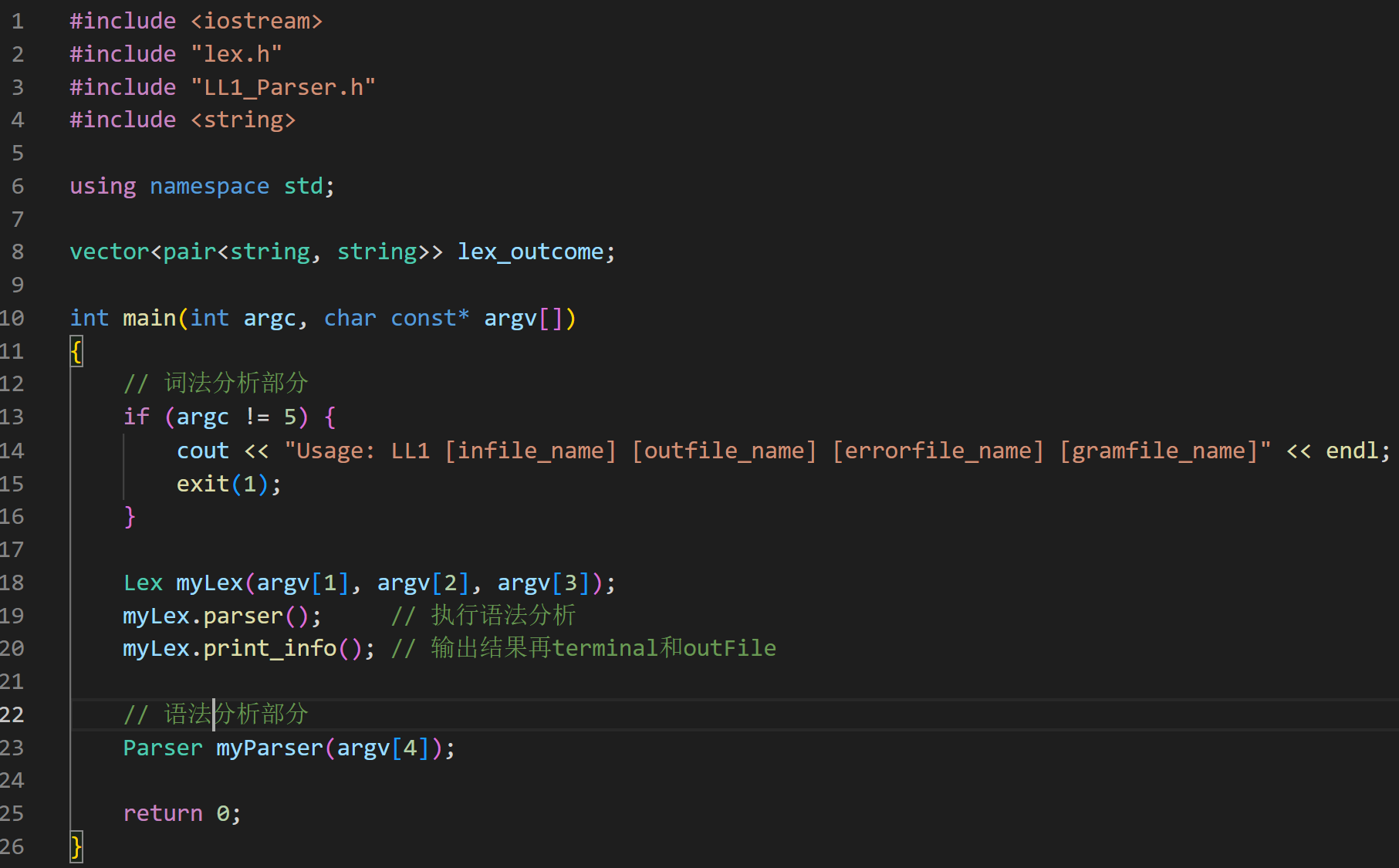
5. Lex\_Error：输出错误词法模块

6. LL1\_Grammar：分析文法，输出分析表

7. LL1\_Parser：分析输入表达式，根据分析表构造分析过程

## 2.3 Main函数

Main函数的主要功能：



1. 获得输入参数，分别为**[infile\_name] [outfile\_name] [errorfile\_name] [grammarfile\_name]**

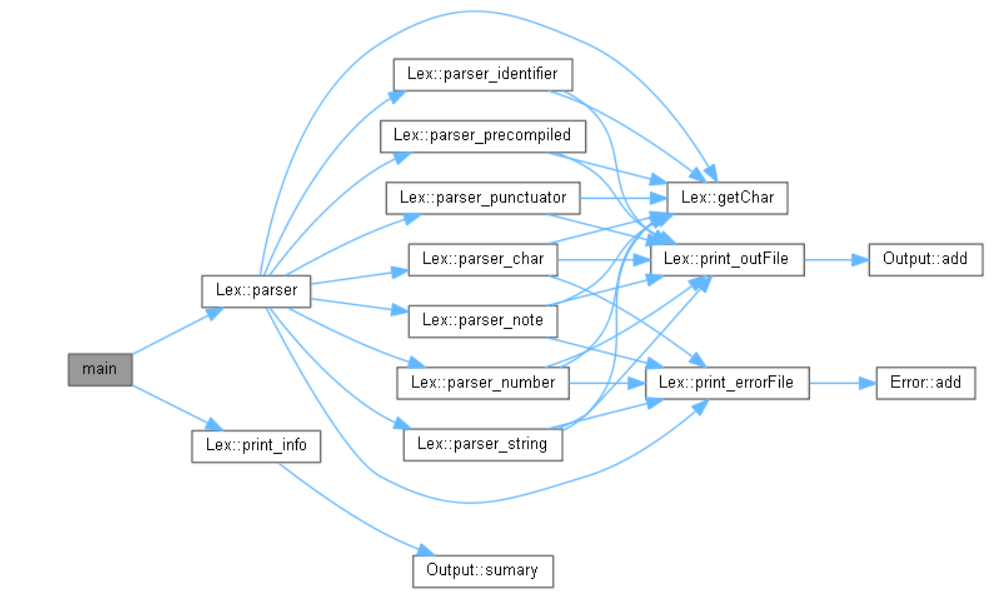
若格式不正确则报错exit(1)。

2. 运行lex类分析相应的inFile。

3. 用lex统计输出的函数输出有关的所有统计信息（同时打印在out和terminal里）。

4. 用Parser来分析程序（Parser内包含Grammar类进行文法分析表的构造，然后进行表达式分析）

生成的拓扑图：



## 2.4 Lex-识别进入各类词法函数

Lex就不进行详细介绍了。

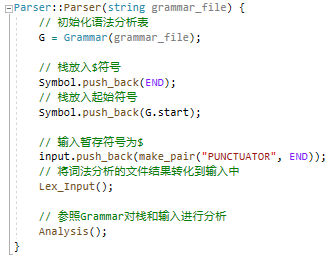
## 2.5 Grammar类详解



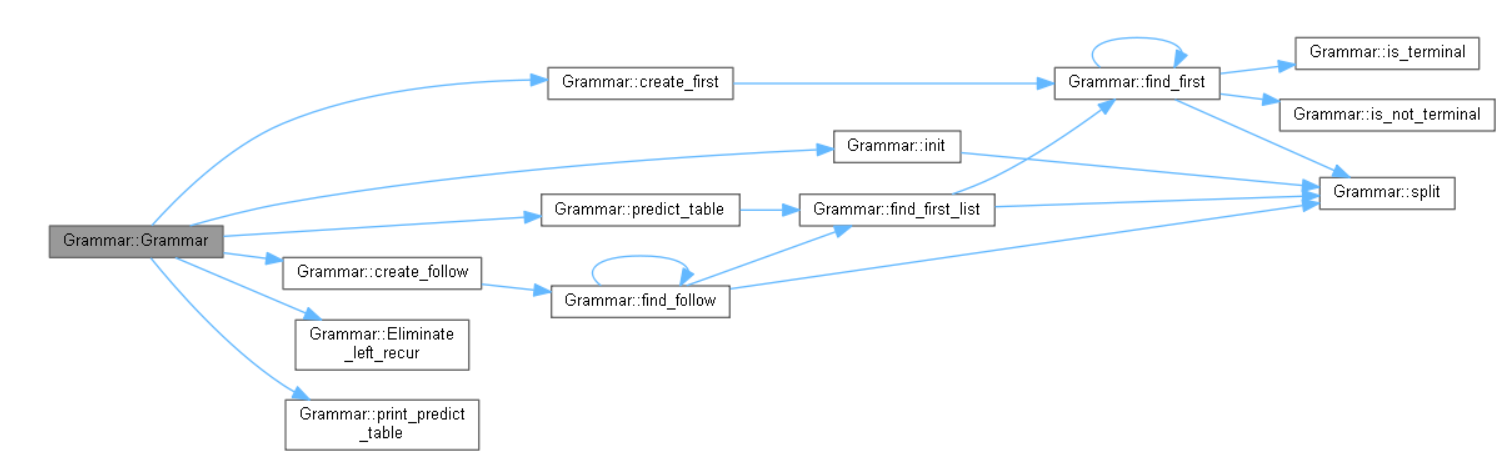
我觉得还是主要重点分析数据类型、我们用**set存符号，用map存firstfollow，用map存预测分析表**。

尤其是预测分析表的构造。我们用**前一个vector所确定非终结符遇到的终结符**，来寻找到对应的须要调用的产生式。这是一个非常巧妙的做法。

根据输入语法来进行语法分析表的生成。其实主要的步骤就是这些：



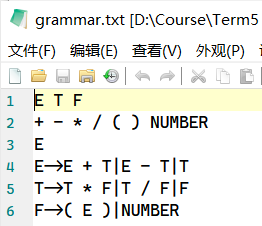
函数调用如下：



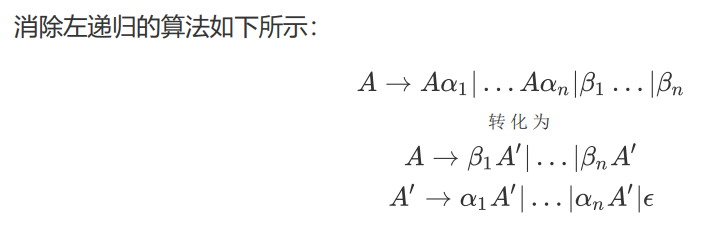
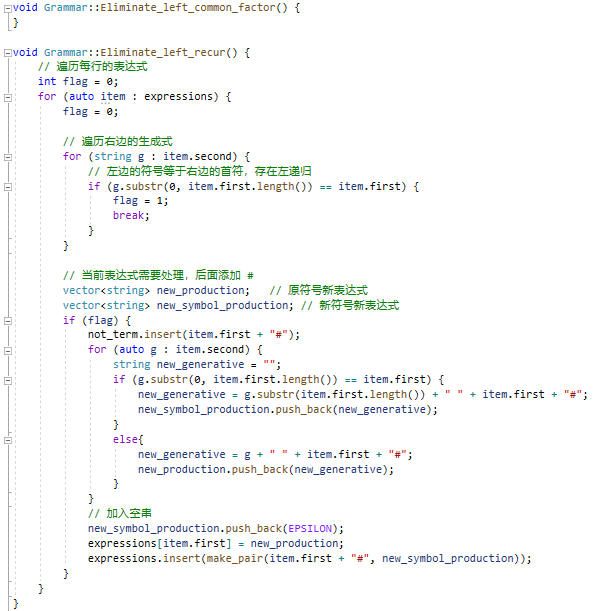
### 2.5.1 Grammar-获得文法输入信息

这里规定文法的输入是这样的：

* + - 第一行是非终结符
    - 第二行是终结符
    - 第三行是起始符
    - 后面是文法表达式，但是文法生成式各符号间一定是空格隔开（正则匹配太离谱了）

### 2.5.2 Grammar-消除左公因子和左递归

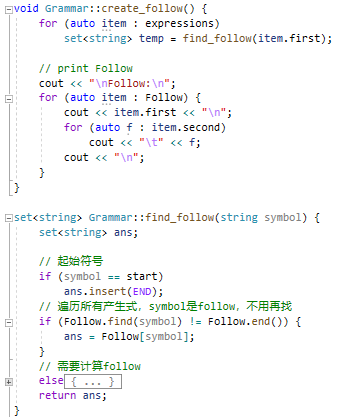
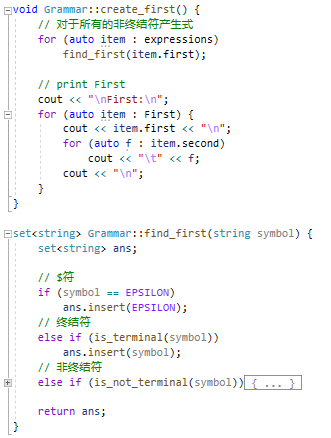
 

实质就是找到左公因子，提取；找到左递归，加入新符号，生成文法表达式。

我们这里新符号就是在原符号后面加入**#（如E变成E#）**

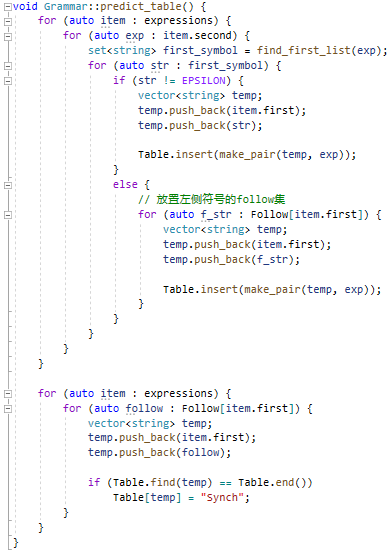
### 2.5.3 Grammar-寻找First和Follow

这里就是按照书上给的递归算法来实现了。我们这里写了对于每一个生成式的递归调用。



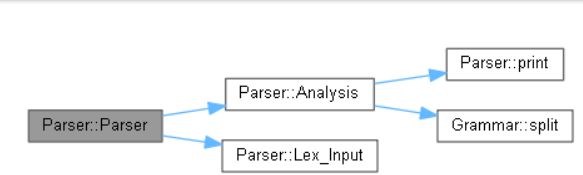
这里要注意的是，first的寻找多了一个函数，是为了计算符号串的first。

### 2.5.4 Grammar-构造文法分析表

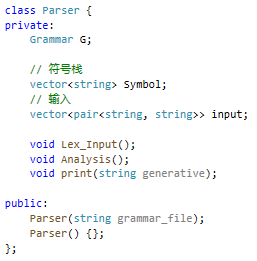


我们**预测分析表的构造**还是挺明确的。就是根据题目列出的算法来实现。这也是**本次实验的重点之一。**

在这个中间，我们也加入了错误处理。但很遗憾没有太多时间进行错误测试。

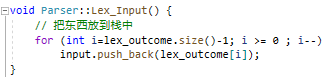


## 2.6 Parser类详解



parser就显得比较简单，其实就是把词法分析结果放到栈里面，边分析边输出。

### 2.6.1 Parser-读入词法分析结果到栈

非常简单，把词法分析pair的vector倒序插入栈。

### 2.6.2 Parser-分析过程

很简单的根据当前字符和输入找文法的过程。

只不过要注意符号序列消除的问题。

### 2.6.3 Parser-输出

我觉得还是很简单

就是填满相应的空格大小输出就可以。

## 2.7 运行结果

First和Follow集，以及预测分析表



分析过程

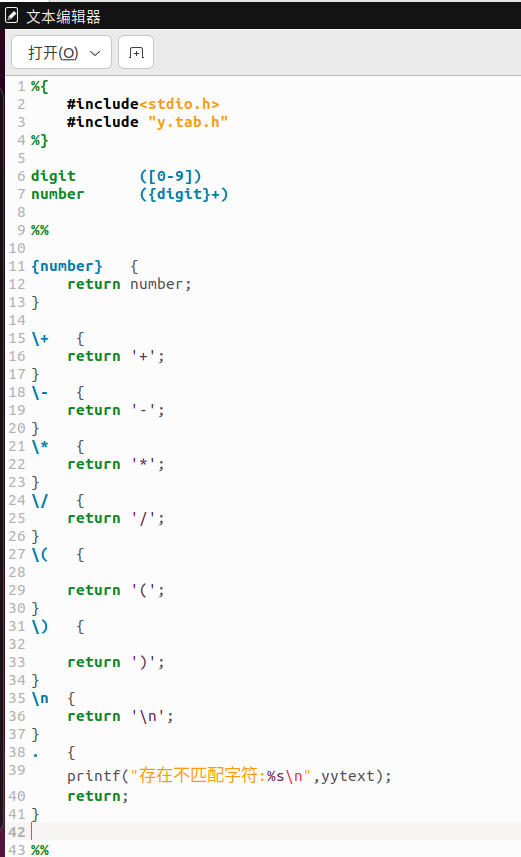


样例程序可以运行完毕。（没有过多测试失败情况和左递归情况）

# 3. YACC

## 3.1 Lex部分

比较简单，根据提供的文法只需识别 +、-、\\*、/、（、）这几个终结符和 num。

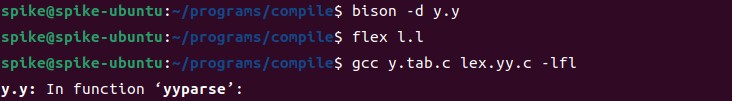


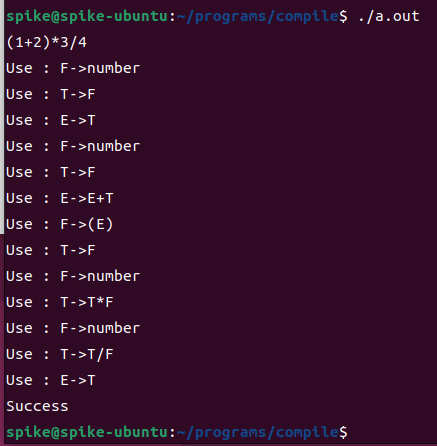
## 3.2 YACC部分

最主要的部分就是非终结符产生式部分。



## 3.3 运行结果





# 4. 问题&解决

## 4.1 Lex->Parser

本来想着直接用文件来读取。因为lex最终把词法分析结果输出到文件。但是在写完读文件之后，一直读的是空文件。可能是因为系统调用的时间比不上运行时间，所以就改用全局变量了。

## 4.2 First和Follow递归

在这之间我尝试了很多错误方法。。。不过还是解决了。 主要出在First的处理上。有时候递归不到位导致少了。还有Follow的$处理一开始也忘了。

## 4.3 栈的分析过程

栈在分析的时候还是有很多需要主义的。尤其是消除的时候，可能是多个符号序列来消除。这时候需要记录前几个。就是在这里花费了我很多时间。

# 5. 实验总结

实现了LL1和YACC

LL1：文法输入，左递归/左公因子的消除，first/follow集合的生成，分析表的构建等，非常难，花了很长时间。但YACC好像还挺简单。这两个一对比发现YACC好太多了。不过这LL1加深了我对课本知识的理解。