**《编译原理与设计》**

# LL(1)语法分析程序

# 的设计与实现

实验报告

班级 2018211308

学号 2018211393

姓名 王一丰

日期 2020.11.14

目录

[LL(1)语法分析程序 1](#_Toc56276985)

[的设计与实现 1](#_Toc56276986)

# 实验内容

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：

E → E+T | E–T | T

T → T\*F | T/F | F

F → (E) | num

在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

编写 LL(1)语法分析程序，要求如下：

(1) 编程实现算法 4.2，为给定文法自动构造预测分析表；

(2) 编程实现算法 4.1，构造 LL(1)预测分析程序。

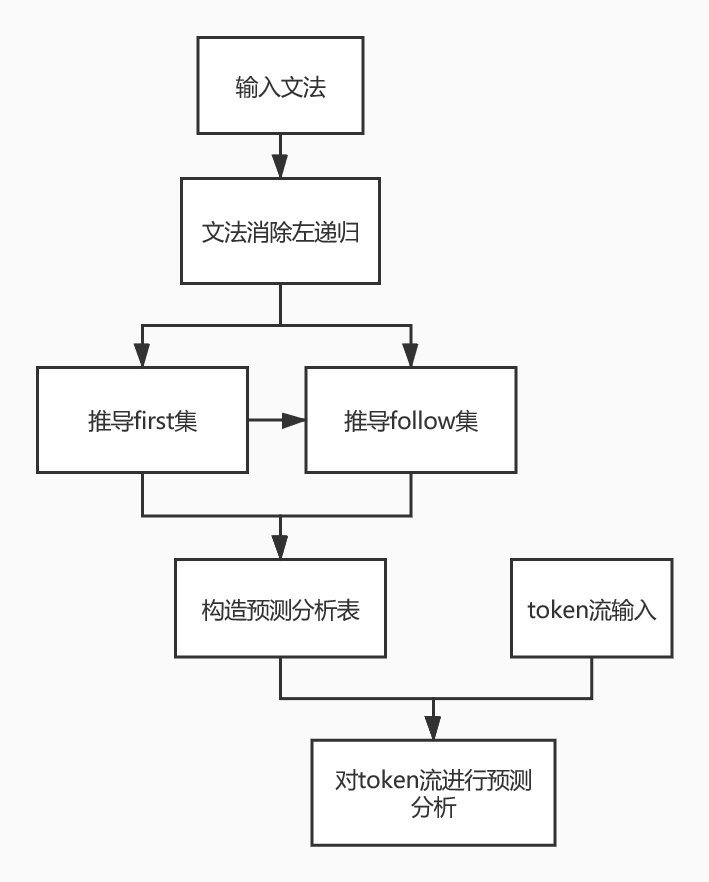
# 二．实验环境

硬件环境：搭载了微软Windows10操作系统的笔记本电脑；

软件环境：Visual Studio 2017 15.8.9集成开发环境。

# 三．实验设计

## 1.总体思路



## 2.数据结构与函数说明

struct production { //产生式数据结构

token left; //左侧非终结符

deque<token> right; //右侧产生式

};

class grammar { //文法类

public:

vector<token> non\_terminators, terminators; //非终结符集，终结符集

token start\_token; //文法起始符

vector<production> productions; //文法所有产生式

map<token, bool> first\_over, follow\_over; //记录非终结符first、follow集是否已求

map<token, set<token>> first,follow; //非终结符的first、follow集

map<token, set<int>> productions\_class; //按产生式左侧非终结符为产生式分类

vector<set<token>> first\_of\_production; //产生式的first集

map<token, map<token, bool>> follow\_include; //follow集的互相包含关系

public:

grammar(); //默认构造函数，通过文件读入文法

void remove\_recursion();//消除左递归

void first\_derivation(const token&);//对应非终结符first推导

void follow\_derivation(const token&);//对应非终结符follow堆推导

void output\_productions();//输出所有产生式

void output\_first();//计算first集

void output\_follow();//计算follow集

void output\_first\_follow();//打印first、follow表

};

class parser //LL(1)语法分析器类

{

private:

grammar g; //分析文法

map<token, map<token, int>> parser\_table;//预测分析表

vector<token> token\_stream;//输入字符流

public:

void input\_token();//通过文件读入字符流

void table\_derivation();//预测分析表推导

void output\_parser\_table();//输出预测分析表

void predict();//预测分析推导

};

# 四．实验原理

## 1.消除左递归：

### 算法原理：

消除左递归算法：

（1） 把文法G的所有非终结符按任一顺序排列，例如，A1，A2，…，An。

（2）for （i＝1；i<=n；i++）

for （j＝1；j<=i－1；j++）

{ 把形如Ai→Ajγ的产生式改写成Ai→δ1γ /δ2γ /…/δkγ

其中Aj→δ1 /δ2 /…/δk是关于的Aj全部规则；

消除Ai规则中的直接左递归；

}

（3） 化简由（2）所得到的文法，即去掉多余的规则。

### 算法实现：

伪代码如下：

遍历所有终结符

{

//消除间接递归

遍历该终结符的所有产生式

If（第一个字符是否已消除左递归）

{

删除在目前非终结符下的产生式

遍历该字符对应的所有产生式

{

原第一个字符的产生式右端插入原生产式第一个字符  
}

}

//判断左递归

遍历该非终结符的所有产生式

{

判断是否含左递归  
}

//消除左递归

If（该非终结符有左递归）

{

遍历该非终结符下所有产生式

{

产生式尾部加入新非终结符  
}

新产生式左侧为新非终结符，右侧为空

遍历所有左递归产生式

{

产生式左端为新非终结符

右端首字符弹出

右端末尾加入新非终结符

}

}

}

## 2.first集

### 算法原理：

对任意文法符号串，其 FIRST集为其可推导出的开头终结符号集合。

构造任意文法符号𝑋的 FIRST集FIRST(𝑋)可遍历𝑋的产生式并利用如下

规则：

(1) 若𝑋 ∈ 𝑉T，则FIRST(𝑋) = {𝑋}；

(2) 若𝑋 ∈ 𝑉N，且有𝑋 → 𝑎…(𝑎 ∈ 𝑉T ⋁𝑎 = 𝜀)，则FIRST(𝑋) = FIRST(X) ∪ {𝑎}；

(3) 若有𝑋 → 𝑌1𝑌2 …𝑌𝑘𝑌𝑘+1 …，且有ε ∈ FIRST(𝑌 𝑖)(𝑖 = 1,…,𝑘)，则 FIRST(𝑋) = FIRST(X) ∪ {𝑎 ∈ FIRST(𝑌 𝑖) | 𝑎 ≠ 𝜀}； 若∀𝑌 𝑖,ε ∈ FIRST(𝑌 𝑖)，FIRST(𝑋) = FIRST(X) ∪ {𝜀}。

### 算法实现：

伪代码如下：

遍历该非终结符对应的所有产生式

{

遍历该产生式右侧字符

{

若为终结符则加入该产生式first集，否则继续遍历

若为非终结符且为求取其first集，则递归求取

若非终结符first集已计算则将其first集加入该产生式的first集

若不含空产生式则结束遍历

若含空产生式，删除first集中的ε，继续遍历

若first集无任何元素加入，则将其first集加入ε

}

将该产生式first集加入所求非终结符的first集

}

## 3.follow集

### 算法原理：

对任意非终结符，其 FOLLOW集是该文法所有句型中紧跟在其后的终 结符或结尾符号 end 的集合。

构造任意文法符号𝑋的 FOLLOW集FOLLOW(𝑋)可遍历文法所有产生式 并利用如下规则：

(1) 若𝑋是起始符号，则FOLLOW(𝑋) = FOLLOW(𝑋) ∪ {end}；

(2) 若有𝐴 → ⋯𝑋𝛽，则FOLLOW(𝑋) = FOLLOW(𝑋) ∪ {𝑎 ∈ FIRST(𝛽) | 𝑎 ≠ 𝜀}；

(3) 若有𝐴 → ⋯𝑋，或有𝐴 → ⋯𝑋𝛽且𝜀 ∈ FIRST(𝛽)，则FOLLOW(𝑋) = FOLLOW(𝑋) ∪ FOLLOW(𝐴)。

由于两个非终结符的 FOLLOW集完全有可能互相包含，此时若在求其 中一个的 FOLLOW直接递归求另一个的 FOLLOW集，则会出现无穷 递归的死循环。引入

follow\_include用于存储非终结符之间FOLLOW集的包含关系，在自己包含对方情况下，先判断对方若也包含自己，则停止递归，在所有推导结束后，互相包含的非终结符FOLLOW集取并集

### 算法实现：

伪代码如下：

遍历非终结符

{

遍历该非终结符对应的产生式

{

遍历产生式右侧所有字符

{

若遍历到该非终结符

{

遍历该非终结符右侧所有字符

{

若为终结符则加入follow集

若为非终结符则将其first集加入该非终结符follow集中

如无空产生式，终止遍历

否则将ε从follow集中删除，继续遍历

}

如果该产生式遍历结束仍有空产生式

{

若该产生式左侧不包含该非终结符

{

将该产生式左侧的follow集加入到该非终结符follow集中

}

}

}  
}

}

}

## 4.预测分析表

### 算法原理：

如有产生式A-> 𝑎,当A呈现于分析栈栈顶时:

1. 如果当前输入符号𝑎∈ FIRST(𝑎)时，𝑎应被选作A的唯一合法代表去执行分析任务，即用𝑎展开A，所以表项M[A,a]中应放入产生式A->𝑎。
2. 如果ε∈ FIRST(𝑎)，并且当前输入符号b∈ FOLLOW(A)，这里b∈ 𝑉T ($),则A-> 𝑎就认为A自动得到匹配，因此，应把产生式A->𝑎放入表项M[A,b]中。

### 算法实现：

伪代码如下：

for(文法G的每个产生式A-> 𝑎)

{

for(每个终结符号a∈ FIRST(𝑎))

把A-> 𝑎放入M[A,b]中;

if(ε∈FIRST(𝑎))

for(任何b∈ FOLLOW (A))

把A-> 𝑎放入M[A,b]中

}

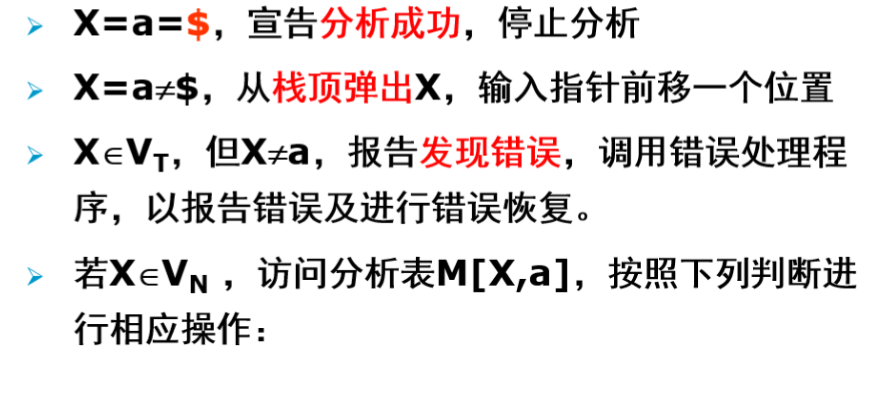
for(所有无定义的M[A,a])标上错误标志

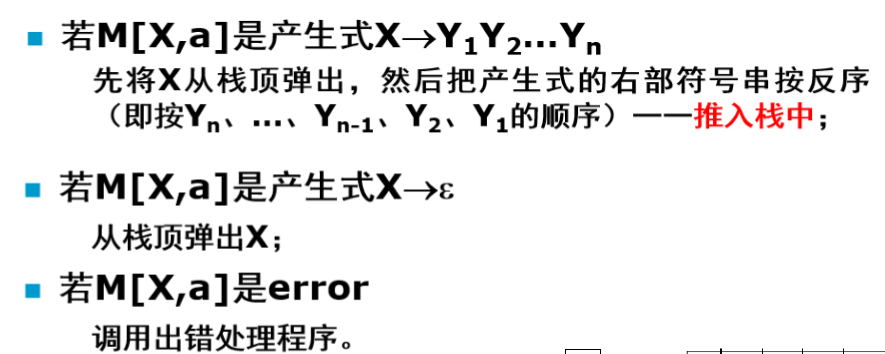
## 5.预测分析程序

### 算法原理：

本部分应用了一个分析表和分析栈，来判断输入字符流，并给出相应的左句型

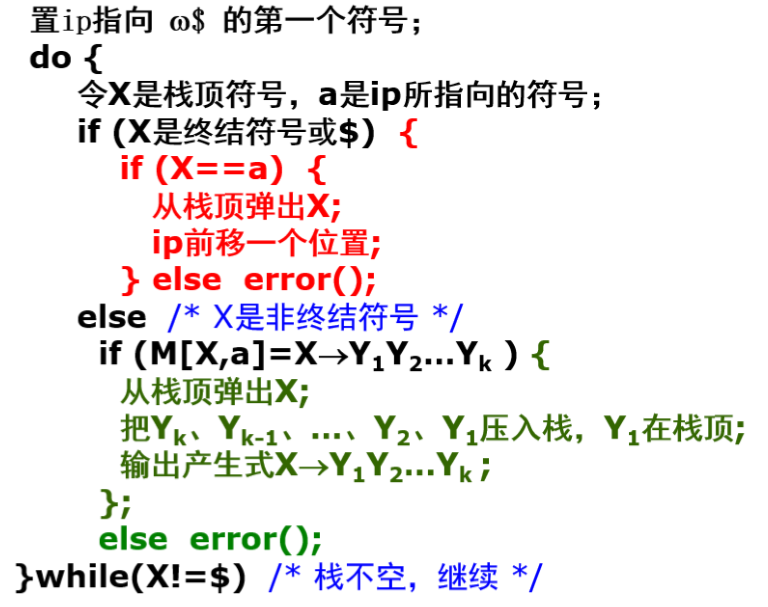
根据栈顶符号X和当前输入符号a，决定分析动作。





### 算法实现：

伪代码如下：



## 6.错误处理

分析过程中,有两种情况可以发现源程序中的语法错误:

-栈顶符号X∈ VT,但X!=a; (a为当前输入的符号)

-栈顶符号X∈ VN,但M[X,a]为空。

错误处理方法:应急式错误处理,使分析工作可以继续进行

-第一种情况,弹出栈顶的终结符号;

-第二种情况,跳过剩余输入符号串中的若干个符号,直到可以继续进行分析为止。

带有同步化信息的分析表的使用

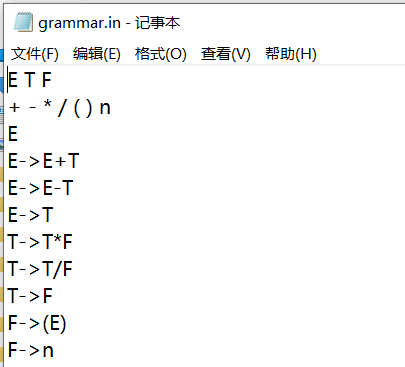
-若M[A,b]为空,则跳过a;

-若M[A,b]为synch,则弹出A.

# 五．测试样例

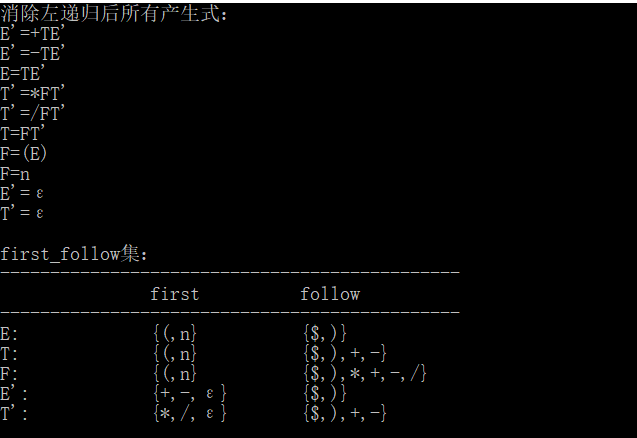
## 1.文法输入

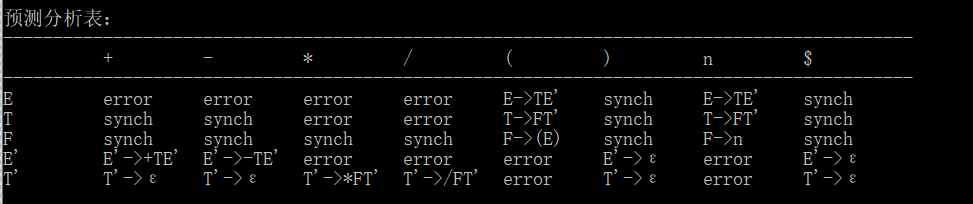
通过grammar.in文件读入所需识别的文法



输入一行非终结符，一行终结符，一行文法开始符，接下来若干行文法的产生式。

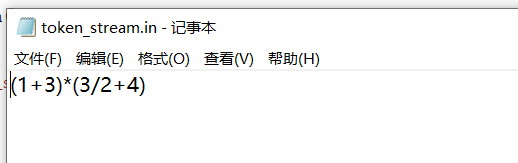
## 2.文法分析结果输出

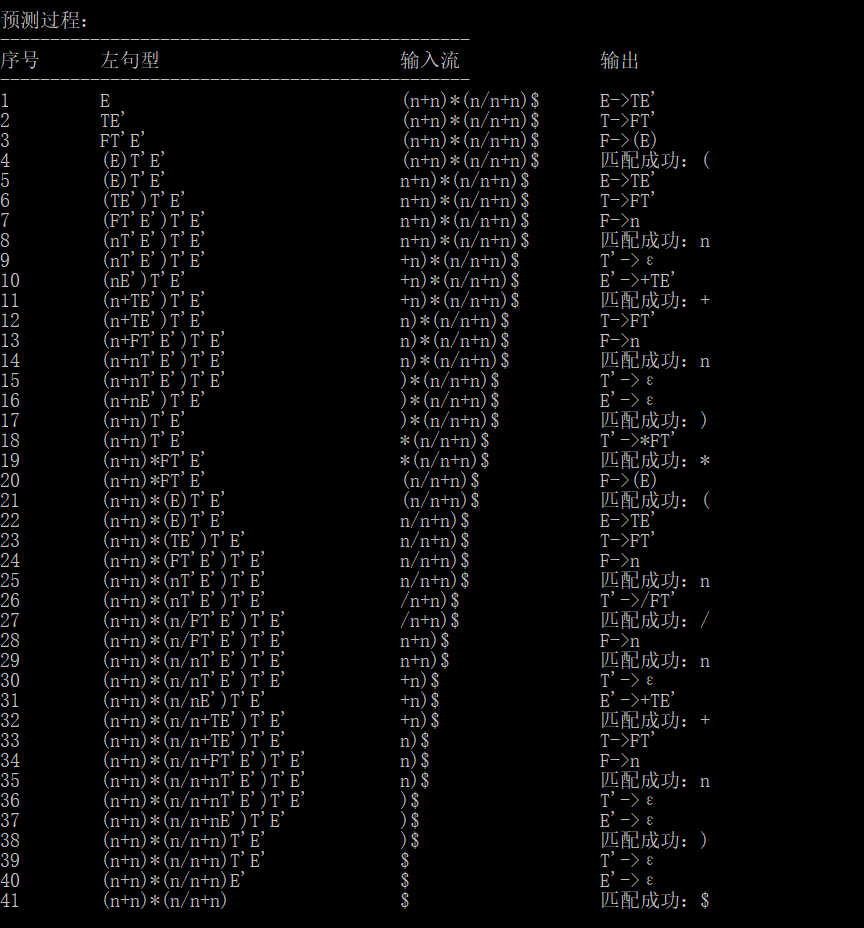


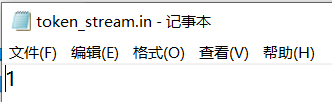


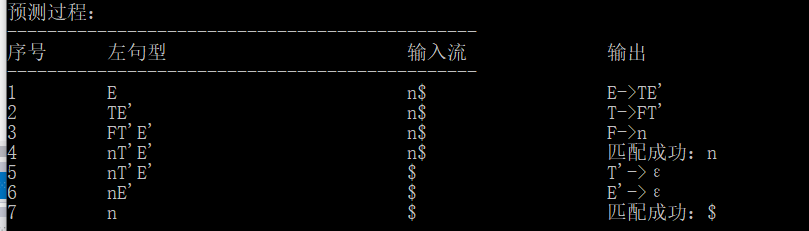
## 3.字符流输入与结果输出

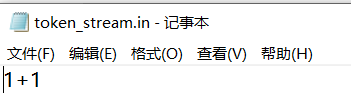
通过token\_stream.in文件读入所要识别的字符流（由于本次实验未链接词法分析器，所以为了不必要的处理，将num更改为n）

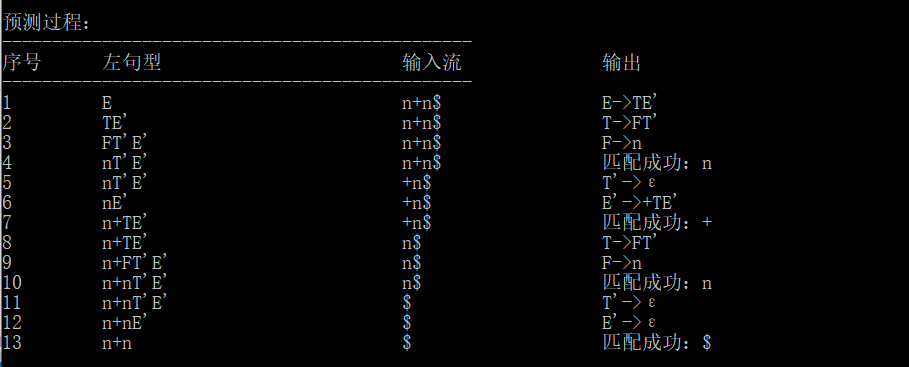


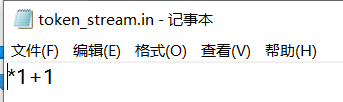


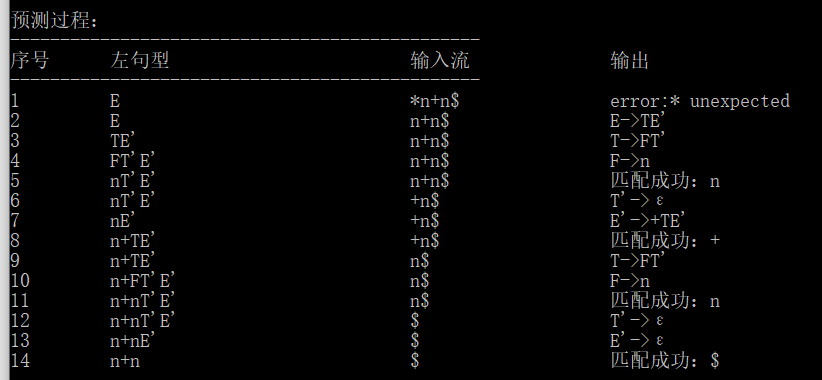


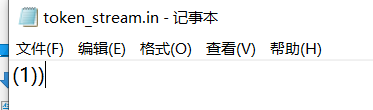


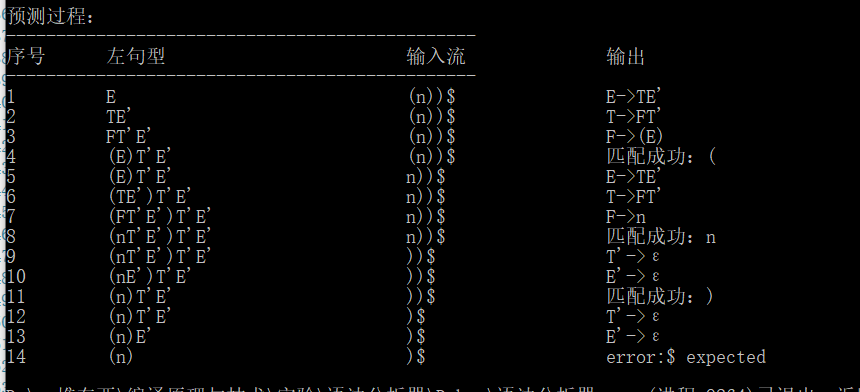












# 六．实验总结

从测试结果可看出，对于正确的表达式，语法分析输出结果也是正确的，而对于错误的表达式，本程序的错误处理措施也能在一定程度上对其进行恢复，当然也具有一定的局限性。

在编译流程中，语法分析是词法分析的后续环节，其分析的基础是词法分析得到的 token流，本次试验中通过将num转换为n，num默认为整数，等手段简化一些不必要步骤，来模拟词法分析器的输入。

编写语法分析程序的过程中遇到了许多，如如何消除左递归，应用怎样的算法消除左公因子，如何解决包含关系，如何解决求取follow集的死循环等问题，这也相当于是对数据结构、算法及编程能力的一个锻炼。