组号： 04

****

编译原理课程设计报告

题 目： 一个简单的Simple-PASCAL编译器

院（系）： 计算机与信息安全学院

专 业： 计算机科学与技术

组 长：

组 员：

指导老师：

职 称：

2021年10月23日

小组分工安排

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 团队成员 | 姓名 | 性别 | 学号 | 分工 | 小组评分 | 工作量（%） |
|  | 男 |  | 词法分析  对应部分报告的编写 |  | 20% |
|  | 男 |  | NFA确定化  DFA最小化  对应部分报告的编写 |  | 20% |
|  | 男 |  | 语法分析  对应部分报告的编写 |  | 20% |
|  | 女 |  | 求First集、Follow集  判断是否是LL(1)文法LL(1)分析表填写  对应部分报告的编写 |  | 20% |
|  | 男 |  | 语义分析  对应部分报告的编写 |  | 20% |

报告评分标准及得分

|  |  |  |
| --- | --- | --- |
| 序号 | 评分标准 | 得分 |
| 1 | 团队组织分工科学合理性（10分） |  |
| 2 | 论文工作量（15分） |  |
| 3 | 论述设计方案与论文组织结构科学性（40分） |  |
| 4 | 报告撰写规范性（15分） |  |
| 5 | 实验结果总结及分析（20分） |  |
| 总分 | |  |

摘 要

编译原理课程设计目的是为了引导学生根据理论实现简单的编译器系统设计，让学生通过自己动手实现简单的编译器具有的词法、语法、语义分析等功能，通过实践对编译程序的设计原理及技术进一步了解和掌握，培养学生计算思维能力、锻炼学生动手以及将理论知识运用于实践的能力。

在本次课程设计中，我们小组设计并实现了一个简单的Simple-PASCAL编译器。该编译器能够对符合类PASCAL文法的源程序进行词法、语法、语义分析并给出分析结果。同时实现有限自动机确定化、最小化算法，通过子集法将NFA确定化为DFA，再通过等价划分法将DFA进行最小化；另外，选择自顶向下语法分析方法，实现求解First、Follow集合、判定文法是否为LL(1)文法及LL(1)分析表填写算法。

在词法分析模块中，设计相应单词的有限自动机并进行化简，再根据简化的有限自动机设计并实现词法分析器，该词法分析器主要通过有限自动机的状态跳转来实现，根据自动机结束状态来得到该单词的TOKEN值。该词法分析器能够分解源程序中的单词，过滤掉空格、Tab和回车、跳过注释，将单词流在屏幕上显示输出并存入指定的文件中，并且能够能对存在的错误进行相应错误类型报错；在语法分析模块中，利用词法分析的结果进行语法分析器设计。该语法分析器采用SLR分析法，根据文法构造SLR分析表，然后编写分析表驱动程序、语法分析程序，输出语法分析过程并检查、报告语法错误；在语义分析模块中，使用S属性翻译文法，通过语法制导翻译完成中间代码生成，把源程序翻译为四元式流。

关键词：PASCAL编译器；词法分析；语法分析；语义分析

目录

[1. 引言 1](#_Toc17416)

[1.1 编译原理概述 1](#_Toc5399)

[1.2 课程设计的意义 1](#_Toc22311)

[1.3 课程设计的内容 1](#_Toc12209)

[2. 课程设计任务及要求 1](#_Toc20117)

[2.1 课程设计任务 1](#_Toc10829)

[2.1.1 词法分析任务 1](#_Toc6184)

[2.1.2 语法分析任务 2](#_Toc3332)

[2.1.3 语义分析任务 2](#_Toc28382)

[2.2 课程设计要求 2](#_Toc21747)

[2.2.1 词法分析要求 2](#_Toc22775)

[2.2.2 语法分析要求 2](#_Toc24210)

[2.2.3 语义分析要求 2](#_Toc15222)

[3. 词法分析程序设计及实现 3](#_Toc13211)

[3.1 字符串及其类型： 3](#_Toc22253)

[3.2 规定字符串类型： 3](#_Toc24873)

[3.3 分割字符串 3](#_Toc24461)

[3.4 报错处理 3](#_Toc24585)

[3.5 输出结果 3](#_Toc13004)

[4. 有限自动机确定化、最小化算法实现 4](#_Toc10318)

[4.1算法思想 4](#_Toc32542)

[4.1.1根据子集构造法对NFA确定化 4](#_Toc4760)

[4.1.2 DFA的化简 4](#_Toc5783)

[4.2设计流程 5](#_Toc31942)

[4.2.1主要流程 5](#_Toc11113)

[4.2.2存储结构 5](#_Toc5338)

[4.2.3确定化流程 6](#_Toc28070)

[4.2.4最小化流程 7](#_Toc1115)

[4.3实现结果 7](#_Toc31652)

[4.4主要函数代码 9](#_Toc16248)

[5. 语法分析程序设计及实现 11](#_Toc25702)

[6. 自顶向下LL(1)语法分析 12](#_Toc29828)

[6.1 系统分析 12](#_Toc15964)

[6.2 数据结构 13](#_Toc21613)

[6.2.1 结构体与变量定义 13](#_Toc20706)

[6.2.2 函数定义 13](#_Toc4919)

[6.3 求取First集 14](#_Toc2570)

[6.3.1 算法原理 14](#_Toc32075)

[6.3.2 算法流程图 14](#_Toc10447)

[6.3.3 算法具体实现 15](#_Toc12574)

[6.4 求取Follow集 16](#_Toc26512)

[6.4.1 算法原理 16](#_Toc17999)

[6.4.2 算法流程图 16](#_Toc4930)

[6.4.3 算法具体实现 17](#_Toc25224)

[6.5 求解Select集 18](#_Toc15691)

[6.5.1 算法原理 18](#_Toc21103)

[6.5.2 算法流程图 18](#_Toc25898)

[6.5.3 算法具体实现 19](#_Toc12084)

[6.6 LL(1)文法判定 20](#_Toc8272)

[6.6.1 算法原理 20](#_Toc7552)

[6.6.2 算法流程图 20](#_Toc14809)

[6.6.3 算法具体实现 21](#_Toc22725)

[6.7 LL(1)预测分析表填写 22](#_Toc20747)

[6.7.1 算法原理 22](#_Toc5707)

[6.7.2 算法流程图 22](#_Toc29888)

[6.7.3 算法具体实现 23](#_Toc25968)

[6.8 运行结果与分析 24](#_Toc27430)

[6.8.1 测试用例 24](#_Toc7908)

[6.8.2 运行结果截图 24](#_Toc4218)

[6.8.3 运行结果分析 27](#_Toc17144)

[7. 语义分析程序设计及实现 27](#_Toc19628)

[7.1四元式结构体定义 27](#_Toc26423)

[7.2主要函数定义 27](#_Toc24333)

[8. 总结 29](#_Toc5596)

[参考文献 30](#_Toc30831)

# 引言

## 编译原理概述

编译原理即是对高级程序语言进行翻译的一门科学技术，完成高级编程语言到机器语言的翻译过程，其核心思想就是把具有同样逻辑结构和思想的一种语言表示的程序转换为另一种语言表示的程序。编译原理课程是计算机专业设置的一门重要的专业课程，旨在介绍[编译程序](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E7%A8%8B%E5%BA%8F/8290180)构造的一般原理和基本方法。内容包括语言和文法、[词法分析](https://baike.baidu.com/item/%E8%AF%8D%E6%B3%95%E5%88%86%E6%9E%90/8853461)、语法分析、[语法制导翻译](https://baike.baidu.com/item/%E8%AF%AD%E6%B3%95%E5%88%B6%E5%AF%BC%E7%BF%BB%E8%AF%91/2665709)、中间代码生成、[存储管理](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E7%AE%A1%E7%90%86/9827115)、代码优化和目标代码生成。

## 课程设计的意义

编译原理课程具有很强的理论性和实践性，编译原理课程设计作为编译原理课程的重要实践，能极大的提高我们的思考能力和动手编程能力，同时也对我们对编译程序的设计原理以及技术进行进一步的理解和掌握以及培养学生的计算思维能力有很大的好处。编译原理课程设计要求学生能够综合运用所学的多门课程的内容来设计并实现一个简单PASCAL语言的编译器，从而巩固和加强编译原理的理论知识，对编译原理的技术有一个更加深刻的认识和理解。

## 课程设计的内容

本次课程设计，使用C++语言、Visual Studio 2019编译软件进行实现，主要分为三个部分进行：词法分析及算法模块、语法分析及算法模块和语义分析模块。不同模块既独立又相互联系，一方面，不同阶段的工作在功能上是相对独立的，另一方面，后一阶段又从它的上一个阶段获取分析结果进行分析，并将结果传递到下一阶段。其中词法分析器分解源程序中的单词，将源程序转化成为一个个的单词符号并以二元式的形式进行输出显示并保存到指定的文件中。语法分析器采用SLR分析法，根据文法构造SLR分析表，然后编写分析表驱动程序，编写语法分析程序，将词法分析产生的二元式结果作为输入，对它进行语法分析。语义分析器使用S翻译属性，采用语法制导方法，即在语法分析的同时进行语义分析，并产生一定的语义动作，来生成中间代码，以四元式的形式进行输出显示。

# 课程设计任务及要求

## 课程设计任务

### 词法分析任务

1. 利用给出的类PASCAL文法，写出符合文法的两个不同源程序样例；设计相应单词的序作为输入，进行词法分析并给出分析结果。
2. 实现有限自动机确定化、最小化的算法，实现求Ƹ闭包的算法。

### 语法分析任务

1. 利用词法分析得到的二元式结果，将源程序用SLR文法进行语法分析程序设计，根据文法构造SLR分析表，然后编写分析表驱动程序，编写语法分析程序,检查源程序是否有语法错误。
2. 实现语法分析阶段的主要算法，选择自顶向下语法分析方法，实现求解First、Follow集合算法，判定文法是否为LL（1）文法的算法及LL（1）分析表填写算法。

### 语义分析任务

根据语法分析采用的策略，编写相应的属性翻译文法，通过语法制导翻译完成中间代码生成，按照通用的四元式格式，设计中间语言，把源程序翻译为四元式流。

## 课程设计要求

### 词法分析要求

1. 将准备好的两个PASCAL 源程序作为输入，分解源程序中的单词，删除无用的空白字符、回车符以及其它非实质性符号，删除注释行，识别出源程序中的各个基本语法单位（单词），按照规则转换成二元式的形式输出单词流在屏幕显示，并存入指定的文件。
2. 能对存在的错误进行报错，错误类型定义准确，对标识符的长度控制在 8个字符（包括8个）以内，超过的做截断处理；整数不大于 65535，否则报错；能够检查单词的拼写错误，并报错；能够识别注释，并去掉注释。
3. 所实现有限自动机确定化、最小化的算法的正确性，对输入的NFA确定化为DFA能得到正确的结果，DFA最小化后能得到正确的结果。

### 语法分析要求

1. 对词法分析得到的二元流进行语法分析错误检查，语法分析要求检查、报告语法错误，要给出语法在源程序中的位置，进行语法错误恢复。
2. 采用SLR分析法：首先根据文法构造SLR分析表，然后编写分析表驱动程序，编写语法分析程序。
3. 所实现算法的正确性，对于输入的文法，要求能够得到正确的First集、Follow集，能够正确判断文法是否为LL(1)并能正确构造出LL(1)分析表。

### 语义分析要求

通过语法制导翻译给出相应的四元式序列。

# 词法分析程序设计及实现

## 3.1 字符串及其类型：

struct mem

{

char\* str; //字符串

int type; //类型

};

其中，str为语句分割后的单词字符串，type为每一个单词字符串对应的词法类型的编码。

标识符 1 1

整数 2 2

关键字 3 3-11

运算符 4 12-21

分隔符 5 22-28

如图所示：



## 3.2 规定字符串类型：

创建类型表，在关键字，运算符，分隔符表中存入对应常用字符。标识符，整数表则由文件输入的Pascal程序决定，读入一个若找不到则存入标识符，整数表并存入字符串及其小类型表中，找到则直接存入字符串及其小类型表中。

## 3.3 分割字符串

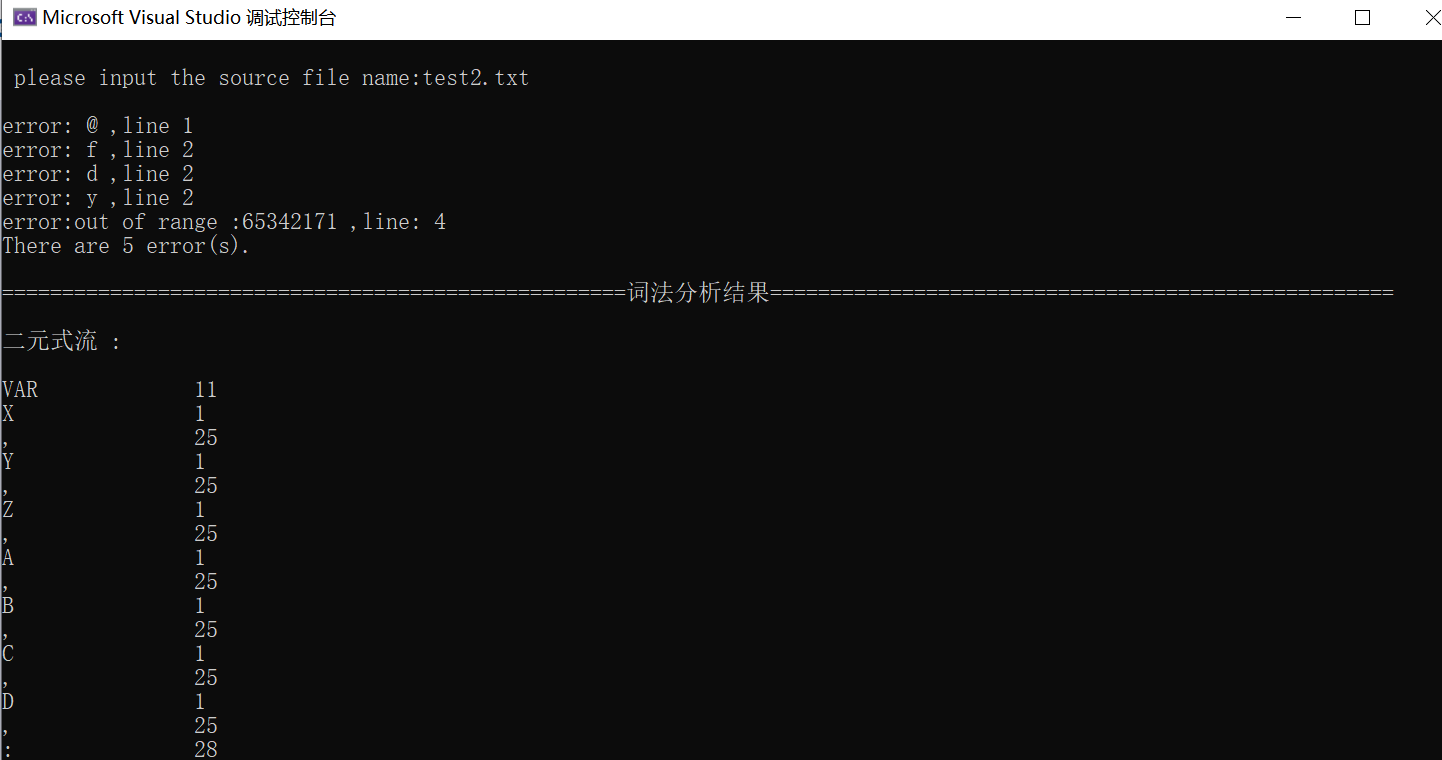
分割字符串，即将文件输入的pascal语言程序分割成类型不同的字符串。依次读入字符，第一个读到整数则寻找整数的结束符，结束符为非数字，读到标识符则标识符的结束符为非英文且非数字，其余各种运算符/分割符的结束符为其后不可能的组合；若是备注//，/\*\*/则不读备注中的内容。读入最后一个字符后输入“\0”。在程序读完后输入“#”表示结束。

## 3.4 报错处理

遇到非法字符，整数过大，则报出字符和错误行数。

## 3.5 输出结果

打印出字符串及其小类型表的内容。



# 有限自动机确定化、最小化算法实现

## 4.1算法思想

### 4.1.1根据子集构造法对NFA确定化

1. 假定I是M’的状态集的子集，定义集合I的ε闭包ε-closure（I）
2. 若s∈I，则s∈ε-closure（I）；若s∈I，则从s出发经过任意条ε弧能够到达的任何状态都属于ε-closure（I）。
3. 假定I是M’的状态集的子集,定义Ia=ε\_\_CLOSURE(J)，其中，J是那些可以从I中某一状态结点出发经过一条a弧而到达的状态结点的全体。
4. 构造一张共有K+1列的表（K为变换个数）该表的首列为ε-closure（X）。检查该行上的所有状态子集，看它们是否已出现在表的第一列，将未曾出现者填入后面空行的第一列。重复上述过程，直至出现在第I+1列上得所有状态子集都已经在第一列上出现。

### 4.1.2 DFA的化简

将终态组和非终态组的Ik(k=a,b……)求出，若所有集合都是组集合的子集，则DFA已最小化。

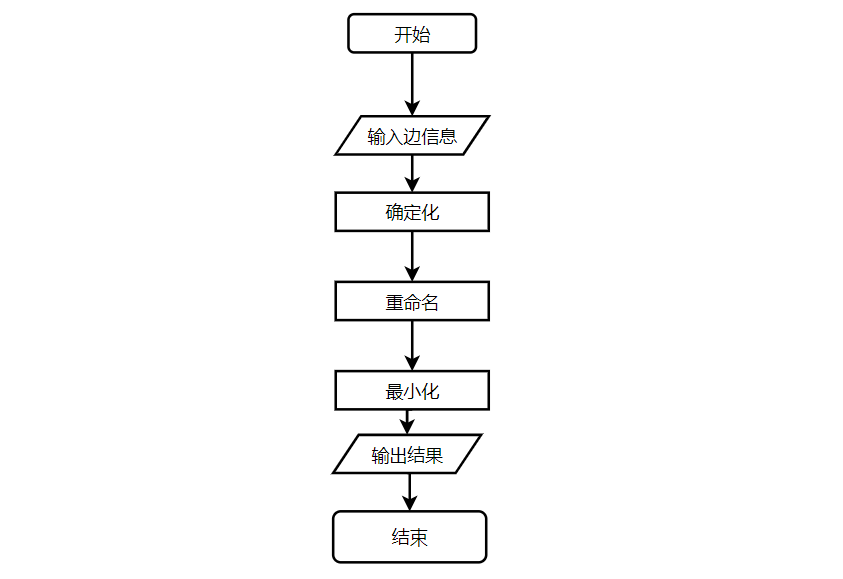
## 4.2设计流程

### 4.2.1主要流程

将边存储在数据结构中，完成对状态的读取，接着开始确定化，接着重命名更改数组内容，接着最小化，最后重命名后输出。主要函数有eclouse，move，output。

### 4.2.2存储结构

(1)边集

struct edge {

string first;//前状态

string change;//运算符

string last;//后状态

};

(2)chan表

struct chan {

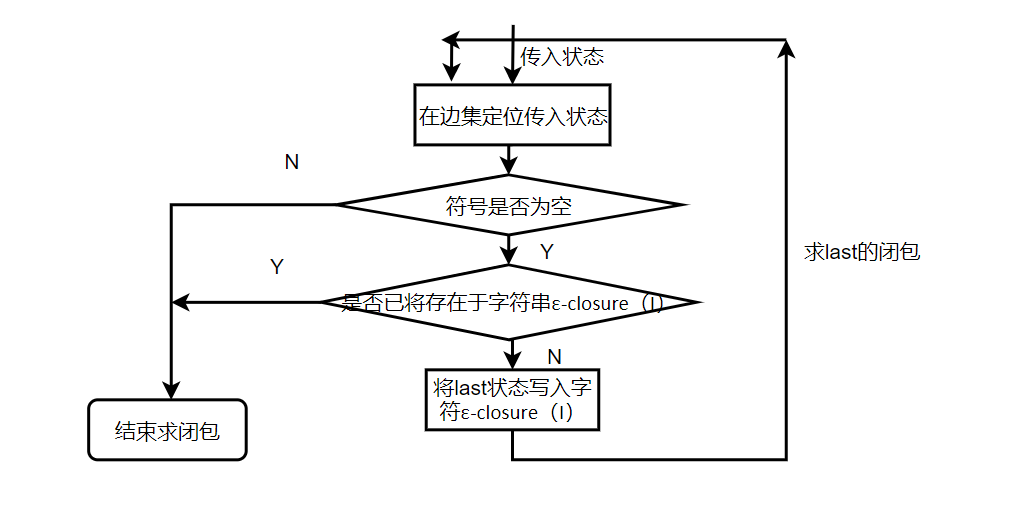
string ltab;//第一列存储状态子集

string jihe[MAXS];//经过非空运算符到达的集合

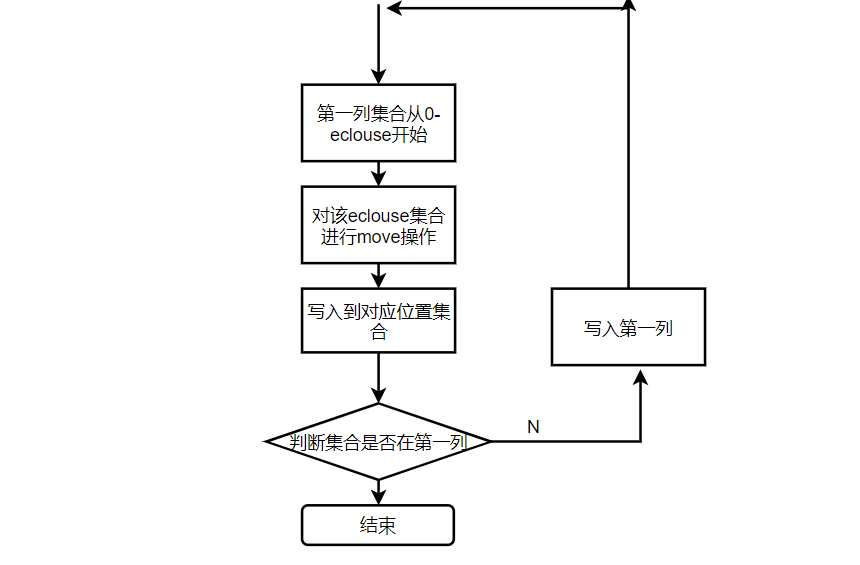
};

### 4.2.3确定化流程

先对状态0求闭包，接着对chan表进行操作。



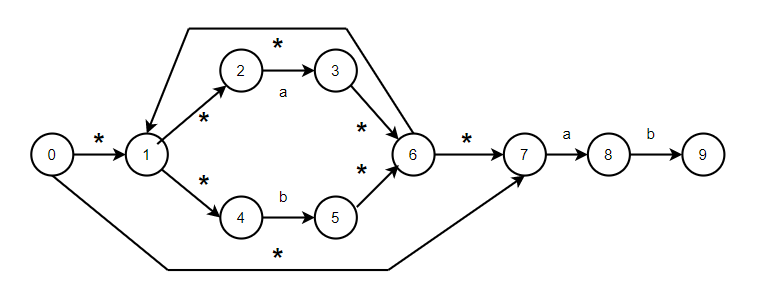
对chan表进行操作



### 4.2.4最小化流程

## 4.3实现结果

测试案例：



输入信息和最终态完成NFA确定化和DFA最小化。

0 \* 1

1 \* 2

1 \* 4

0 \* 7

3 \* 6

5 \* 6

2 a 3

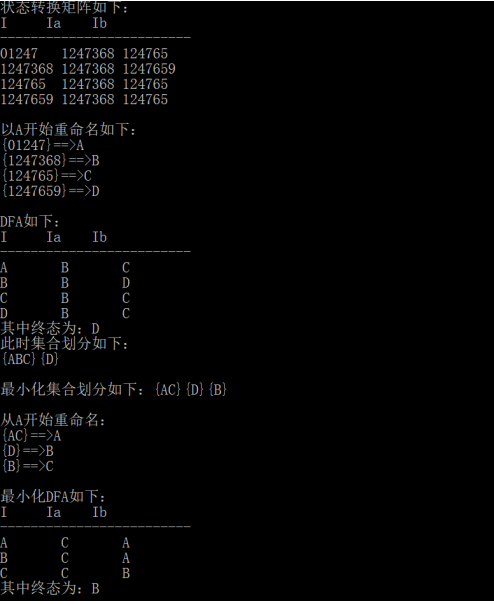
4 b 5

6 \* 1

6 \* 7

7 a 8

8 b 9



## 4.4主要函数代码

(1)void eclouse(char c, string& he, edge b[]) {//求闭包，在字符串中找c：eclouse(b[0].first[0], t[0].ltab, b)

int k;

for (k = 0; k < N; k++) {//循环边个数

if (c == b[k].first[0])

if (b[k].change == "\*") {//如果为符号为\*,记录他的下一个状态

if (he.find(b[k].last) > he.length())//last在he中的位置和he的长度比较？：有无此字符串

he += b[k].last;//记录进he字符串

eclouse(b[k].last[0], he, b);//递归寻找last的\*闭包

}

}

}

(2)//对chan状态表进行move（he，m）找出

void move(chan& he, int m, edge b[]) {

int i, j, k, l;

k = he.ltab.length();//第一列状态

l = he.jihe[m].length();//新

for (i = 0; i < k; i++)

for (j = 0; j < N; j++)//最内层循环边

if ((CHANGE[m] == b[j].change[0]) && (he.ltab[i] == b[j].first[0]))//运算符相同，不在jihe[m]中的

if (he.jihe[m].find(b[j].last[0]) > he.jihe[m].length())//不在jihe[m]中

he.jihe[m] += b[j].last[0];//加入jihe[m]

for (i = 0; i < l; i++)

for (j = 0; j < N; j++)//最内层循环边

if ((CHANGE[m] == b[j].change[0]) && (he.jihe[m][i] == b[j].first[0]))//已经再集合中的

if (he.jihe[m].find(b[j].last[0]) > he.jihe[m].length())

he.jihe[m] += b[j].last[0];//

}

(3) //将终态组和非终态组的Ik(k = a, b……)求出，

//若所有集合都是组集合的子集，则DFA已最小化。

m = 2;

sta.erase();

flag = 0;

for (i = 0; i < m; i++) {

for (k = 0; k < len; k++) {

y = m;

for (j = 0; j < d[i].length(); j++) {

for (n = 0; n < y; n++) {

if (d[n].find(t[NODE.find(d[i][j])].jihe[k]) < d[n].length() || t[NODE.find(d[i][j])].jihe[k].length() == 0)

{//找出子集为空或者不在该组内子集的d[i][j]

if (t[NODE.find(d[i][j])].jihe[k].length() == 0)

x = m;

else x = n;

if (!sta.length()) {

sta += x + 48;//ascill

}

else if (sta[0] != x + 48) {

d[m] += d[i][j];//分出此j状态

flag = 1;

d[i].erase(j, 1);//清楚前面此j状态

j--;

}

break;

}

}

}

if (flag) {

m++; flag = 0;

}

sta.erase();

}

}

# 语法分析程序设计及实现

## 5.1 SLR文法的构造

5.1.1 SLR原文法

源程序设计语言 G[<程序>]

<程序>→<变量说明><BEGIN> <语句表> <END>.

<变量说明>→VAR<变量表>:<类型>；|<空>

<变量表>→<变量表>,<变量>|<变量>

<类型>→INTEGER

<语句表>→<语句> | <语句>;<语句表>

<语句>→<赋值语句>|<条件语句>|<WHILE语句>|<复合语句>

<赋值语句>→<变量>:=<算术表达式>

<条件语句>→IF<关系表达式>THEN<语句>ELSE<语句>

<WHILE语句>→WHILE<关系表达式>DO<语句>

<复合语句>→BEGIN<语句表>END

<算术表达式>→<项>|<算术表达式>+<项>|<算术表达式>-<项>

<项>→<因式>|<项>\*<因式>|<项>/<因式>

<因式>→<变量>|<整数>|(<算术表达式>)

<关系表达式>→<算术表达式><关系符><算术表达式>

注释：从接连出现的/\*到下一次接连出现的\*/之间的任何文字都是注释。

从某行接连出现的//到该行的结尾的任何文字都是注释。

白空格：两个单词之间的任何空格，制表符，回车，换行等都是白空格，除了用来分隔单词以外，没有意义。

限制条件：标识符的长度不大于8，整数不大于65535。

5.1.2 构造SLR文法

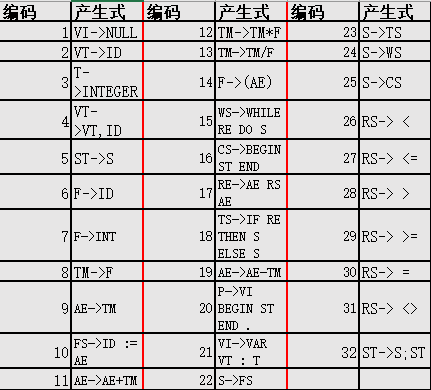
（1）将文法的产生式左部的非终结符单词化及编码，如表4-1所示；

表4-1 非终结符分类及编码



1. 将文法的产生式编码，如表4-2所示；

表4-2 产生式编码



（3）构造语法树

由于涉及部分产生式涉及递归，所以需要具体问题具体分析。下面给出一程序，根据前面得到的产生式语法构造语法树，模拟其语法分析过程。

假设程序如下程序：

VAR

X,Y,Z,A,B,C,D:INTEGER;

BEGIN

WHILE (A+1)>B DO

IF C>D THEN

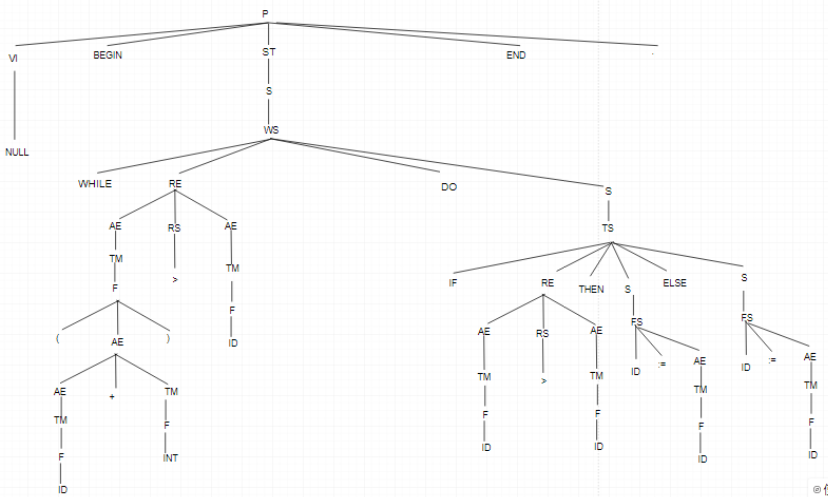
X:=Y+Z

ELSE

Y:=X+Z

END.

语法树构造如下图所示：

图 4-1 语法树模拟语法分析过程

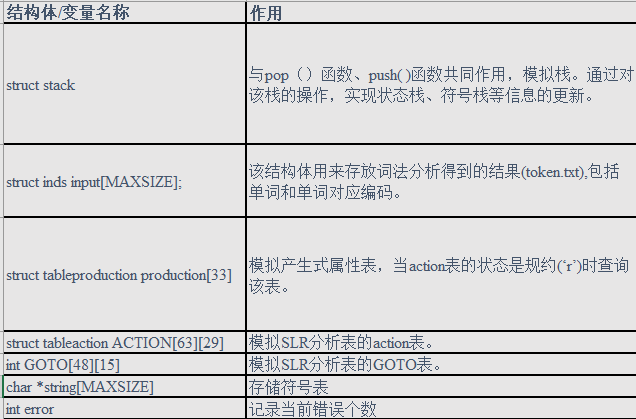
（4） 构造DFA（见附录一）

（5） SLR文法（见附录二）。

5.2 语法分析程序设计与实现

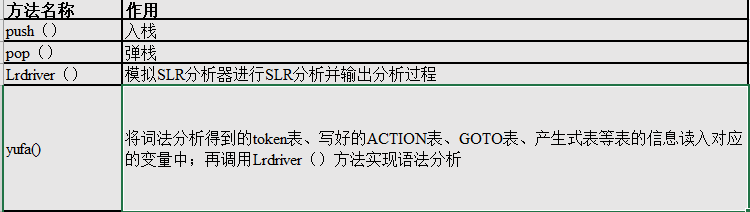
5.2.1设计结构体和主要变量（见表4-3）

表4-3 结构体和主要变量表



4.2.2 主要方法及作用（见表4-4）

表4-4 主要方法和作用表



## 5.3 主要模块的算法

（1）语句存放数组

考虑到程序内的单词是需要依次读取的，为了提高我们的提取效率，这里选择使用一个数组来存放这些单词和对应的编码。考虑到目标程序内的单词数量不确定，我们尽可能让数组的最大长度大一些，用了500。

#define MAXSIZE 500

struct inds

{

char \*instr; //字符串

int sty; //类型

};

struct inds input[MAXSIZE];

（2）SLR分析表的存储结构

struct tableaction //构造action分析表

{

char c; //动作

int s; //状态

}ACTION[63][29]={0};

int GOTO[48][15]={0}; //构造goto表

（3）语句合法性判断

根据 当前栈顶元素内的存储状态变量的值 和 余留串当前指向的单词对应编码 作为ACTION[X][Y]的X、Y坐标，查询ACTION[X][Y]的动作：

a. 若状态是“A”，则接受，SLR分析结束。

b. 若状态是“error”（即空值）则输出报错信息。

c. 若状态是“S”则根据ACTION[X][Y]的状态进行对应的入栈操作。

d. 若状态是“R”，则根据ACTION[X][Y]的状态为下标查询产生式表，根据对应产生式的成员word\_count值将栈回退word\_count位，同时获得其word\_number值（左边产生式的编码值），与回退后的栈顶元素的状态变量的值为下标查询GOTO表，由左边产生式的编码值、GOTO表的对应内容获得下一步信息，进而执行入栈操作。

下面举例子说明，假设程序前六个单词为“var X , Y , Z”。

首先将单词存入input[ ] 数组，并初始化栈（加入#），如图4-2所示

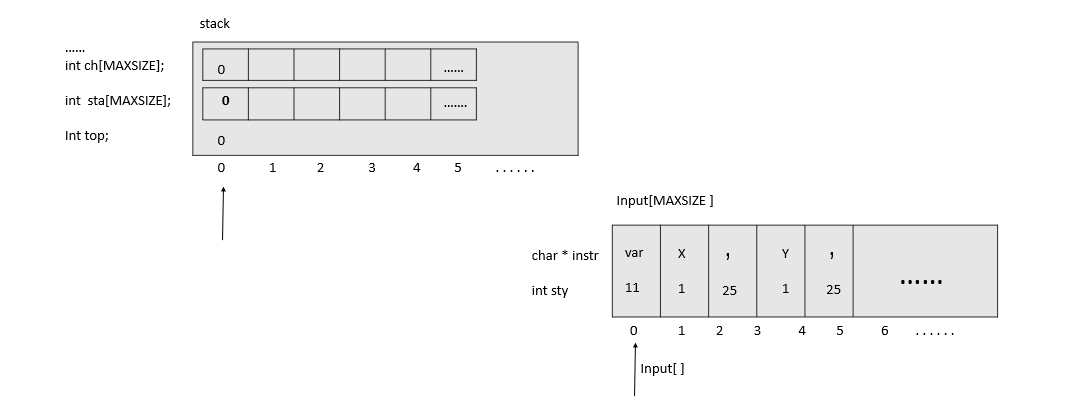


图4-2 初始化栈

这时，根据栈stack的元素sta[0] 、和input[ 0] 的元素sty的值作为action表坐标查action 表，此时即为action [ 0] [ 11]。查表发现，action表该坐标下状态是“S”，故进行入栈操作，结果如图4-3所示。

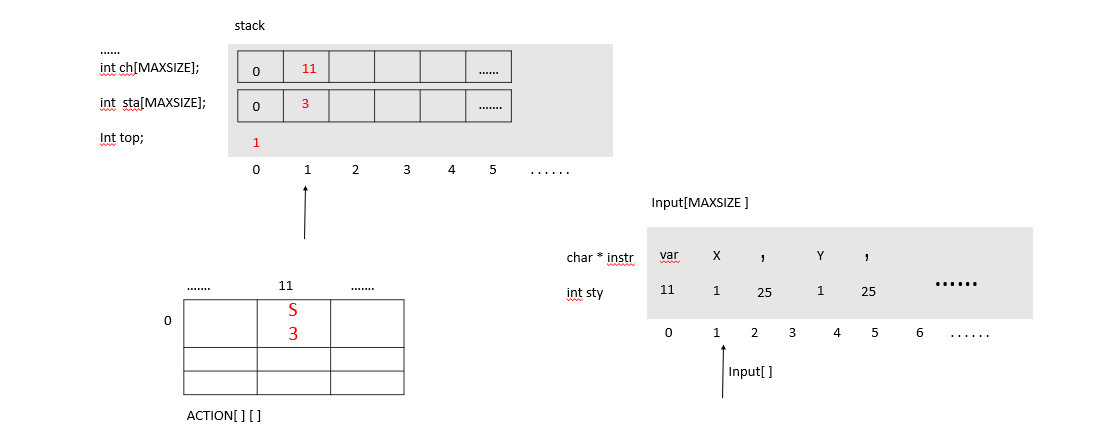


图4-3 移进1

这时，栈中的状态栈就是03，符号栈就是011，栈顶元素为1。下一步对单词“，”进行操作，过程类似上一步，在此不再赘述，结果见图4-4。

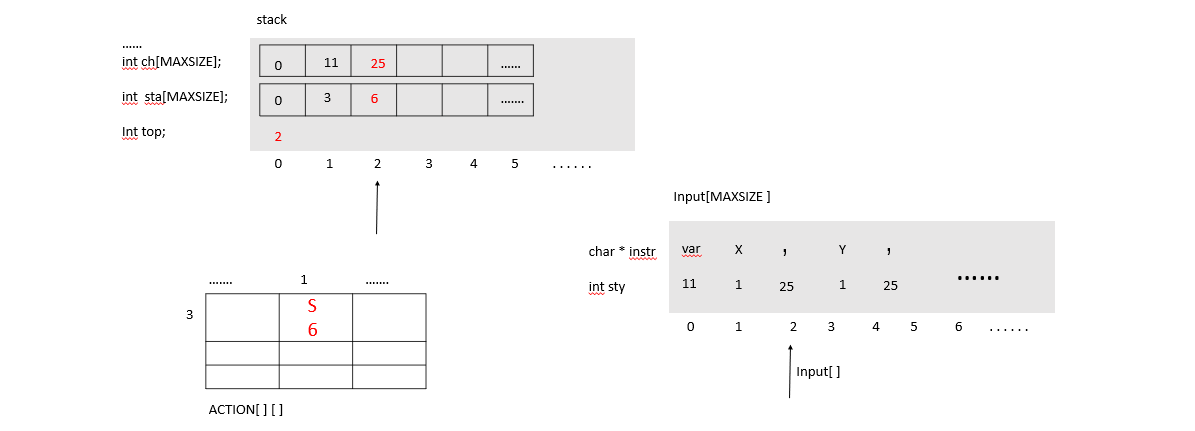


图4-4 移进2

这时，栈中的状态栈就是036，符号栈就是01125，栈顶元素为2。

下一步对单词“Y”进行操作。查询发现ACTION表该坐标下状态是“r”,所以要进行规约操作，先由ACTION表该坐标下的状态值查询产生式表，获取需要弹栈个数和GOTO表的纵坐标，其中，产生式表下的word\_count的值即为栈需要弹栈的元素个数。弹栈后以当前栈的状态栈为GOTO表横坐标，即可查询GOTO表，得到下一步的状态。最后进行入栈操作。入栈后栈中的状态栈就是035，符号栈就是01131，栈顶元素为2，详情见图4-5 。

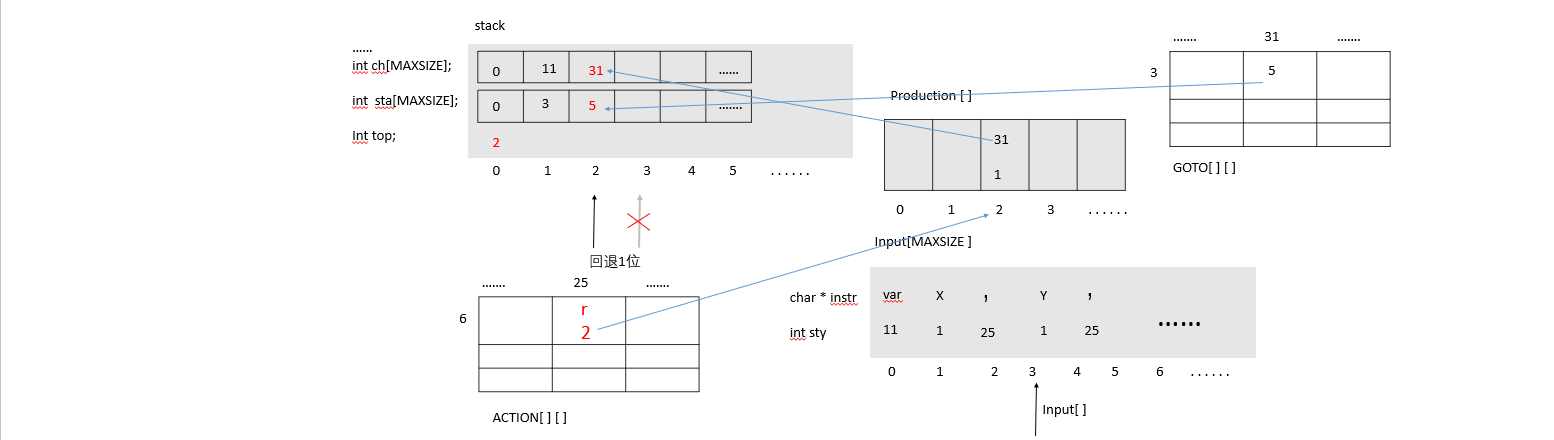


图4-5 规约

以上便是模拟SLR分析表的移进、规约的实现过程。下面是相关代码：

void LRdriver(stack\* st) //SLR分析器驱动

{

int l;

TS = st->sta[st->top]; //栈顶状态号（TopStat）

IS = input[intop].sty; //当前输入单词的编码(InpSym)

while ((ACTION[TS][IS].c != 'A') && (!error))//A为acc

{ /\*输出分析过程\*/

printf("\n");

for (l = 0; l <= st->top; l++)

printf("%d-", st->sta[l]);

if (l >= 12)

printf("\t");

else if (l >= 9 && l <= 11)

printf("\t\t");

else if (l >= 7 && l <= 8)

printf("\t\t\t");

else if (l >= 4 && l <= 6)

printf("\t\t\t\t");

else

printf("\t\t\t\t\t");

for (l = 0; l <= st->top; l++)

printf("%d-", st->ch[l]);

if (l >= 10 && l <= 20)

printf("\t\t");

else if (l >= 7 && l <= 9)

printf("\t\t\t");

else if (l >= 4 && l <= 6)

printf("\t\t\t\t");

else

printf("\t\t\t\t\t");

for (l = intop; input[l].sty != 100; l++)

printf("%-s", input[l].instr);

if (ACTION[TS][IS].c == '\0') //分析动作为ERROR

{

error++;

}

else if (ACTION[TS][IS].c == 'S') //分析动作为移进（s）

{

// 当前单词 单词对应编码 下一步状态

push(st, input[intop].instr, input[intop].sty, ACTION[TS][IS].s, 0, 0, 0);

//栈指向下一个（即可读长度加1）

intop++;

color(FOREGROUND\_INTENSITY | FOREGROUND\_GREEN);

printf("\ts%d", ACTION[TS][IS].s); //输出 : s\*

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

}

else if (ACTION[TS][IS].c == 'r') //分析动作为归约（r）

{

//弹栈（弹栈元素个数由产生式表的属性：word\_count决定）

pop(st, production[ACTION[TS][IS].s].word\_count);

PL = production[ACTION[TS][IS].s].word\_number;//以动作表当前状态值为下标去查产生式表，得到产生式左边的单词对应的值

// 当前状态栈值 产生式左边单词对应的值

NS = GOTO[(st->sta[st->top])][production[ACTION[TS][IS].s].word\_number - 29];//查goto表 ，

// 单词 状态

push(st, input[intop].instr, PL, NS, production[ACTION[TS][IS].s].Pro\_place, production[ACTION[TS][IS].s].Pro\_TC, production[ACTION[TS][IS].s].Pro\_FC);

color(FOREGROUND\_INTENSITY | FOREGROUND\_RED);

printf("\tr%d", ACTION[TS][IS].s);

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

}

TS = st->sta[st->top];

IS = input[intop].sty;

}

if (!error)

{

printf("\n");

for (l = 0; l <= st->top; l++)

printf("%d", st->sta[l]);

printf("\t\t");

for (l = 0; l <= st->top; l++)

printf("%d", st->ch[l]);

printf("\t");

for (l = intop; input[l].sty != 100; l++)

printf("%s", input[l].instr);

color(FOREGROUND\_INTENSITY | FOREGROUND\_RED);

printf("\tAccepted!\n");

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

}

else {

color(FOREGROUND\_INTENSITY | FOREGROUND\_RED);

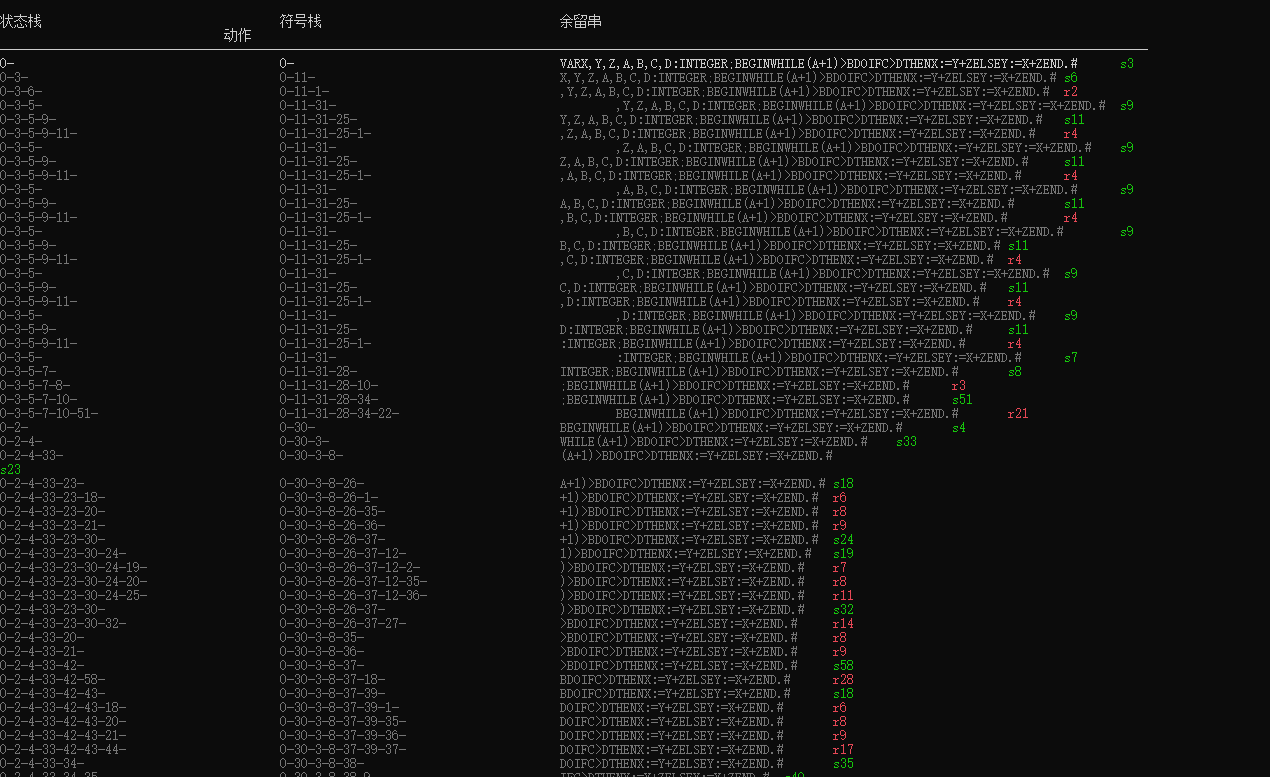
printf("\nIt must be an error here,Please check it again!\n");

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

}

};

## 5.4 运行结果展示

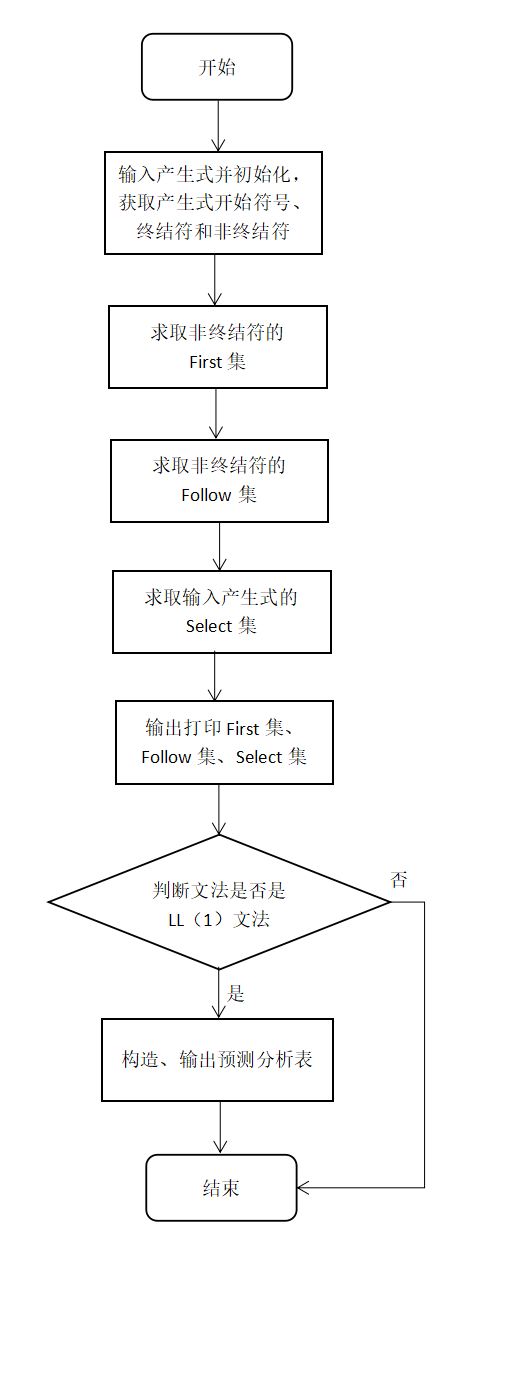


# 自顶向下LL(1)语法分析

## 系统分析

采用自顶向下的分析方法时，必须判别所给文法是否是LL(1)文法，因此对任意给定的文法，都需要先计算出First集、Follow集、Select集，进而判别文法是否是LL(1)文法。在此次LL(1)语法分析程序设计中，输入的文法默认已经消除了左递归（左公因子）。对文法求出了Select集之后，LL(1)文法的判定可以通过判别LL(1)文法的充分必要条件来实现，即对每个非终结符的两个不同的产生式，其求取的Select集合两不相交，即它们的交集为空集。若文法是LL(1)文法，则对其构造并输出预测分析表，否则程序直接结束。实现该算法的系统流程图如图6.1所示。

图6.1 程序系统流程图



## 数据结构

### 结构体与变量定义

//产生式结构体，用来存储产生式左部和右部变量。

struct node

{

char left = '0';

string right;

};

int T; //记录产生式个数

node analy\_str[1000]; //用来存储输入的文法

vector<char> ter\_copy; //用来存储去$终结符

vector<char> ter\_colt; //用来存储终结符终结

vector<char> non\_colt; //用来存储非终结符

set<char> first\_set[100]; //用来存储First集

set<char> follow\_set[100]; //用来存储Follow集

set<char> select\_set[100]; //用来存储Select集

int tableMap[100][100]; //用二维数组存储LL(1)预测分析表

### 函数定义

bool isNotSymbol(char c); //判定给定字符是否是非终结符

int get\_index(char target); //获得在终结符集合中的下标

int get\_nindex(char target); //获得在非终结符集合中的下标

void get\_first(char target); //得到First集合

void get\_follow(char target); //得到Follow集合

void get\_select(); //求取Select集

bool judge\_LL1(); //判定输入文法是否为LL1文法

void input\_initial(); //接收输入文法，处理得到First、Follow集

void display(); //显示First/Follow/Select集与LL1判别结果

void get\_table(); //得到预测分析表

void print\_out(); //输出预测分析表结果

void getAns(); //综合处理，调用各模块的函数

## 求取First集

### 算法原理

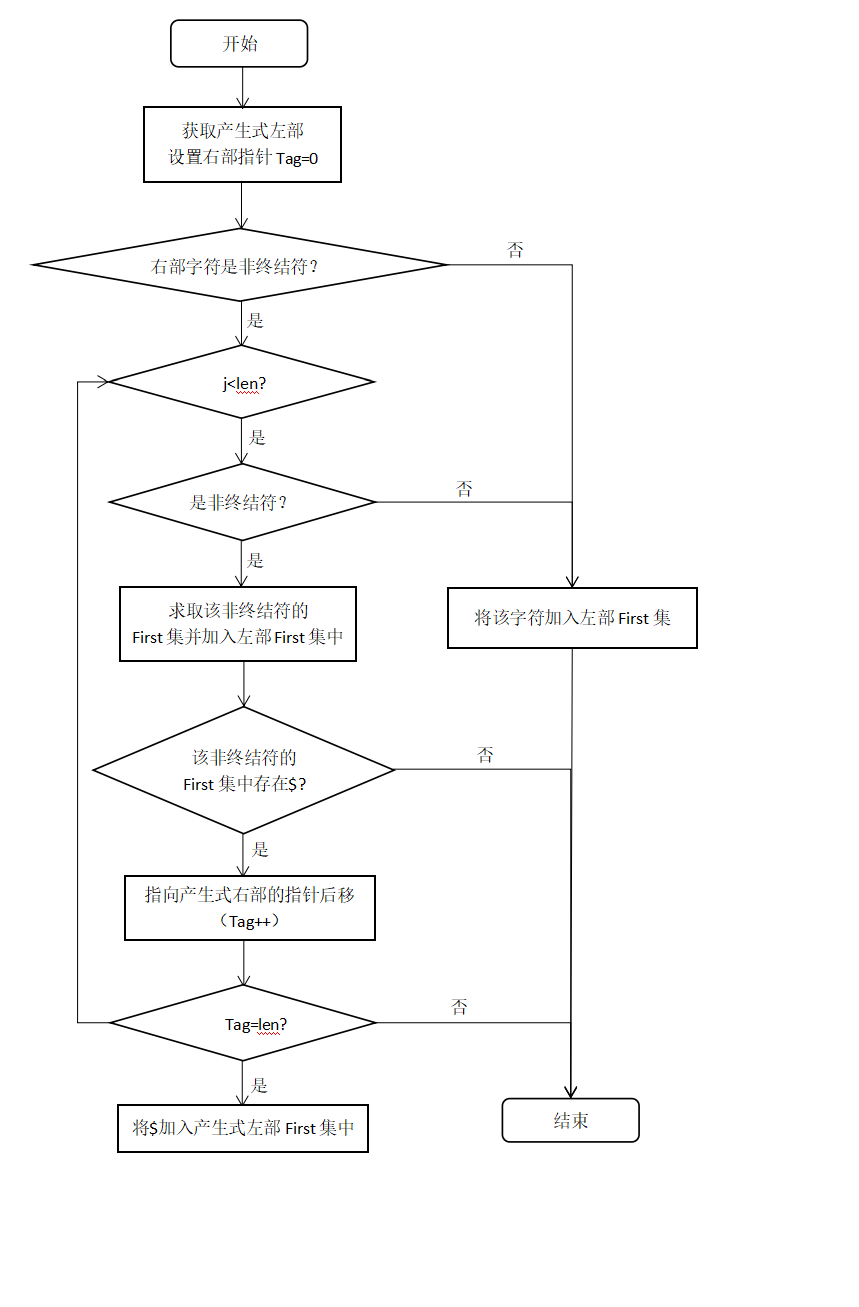
求取文法中非终结符的First集算法原理如下：

1. 若产生式右部第一个字符是终结符，则将其放入左部非终结符的First集中；
2. 若产生式右部第一个字符为非终结符，求该非终结符的的First集；
3. 将去$(用$代替ε)的此非终结符的First集加入左部非终结符的First集中；
4. 若该非终结符的First集中含$，则将指向产生式的指针右移，重复步骤(2)(3)(4)
5. 若该非终结符的First集中不存在$，则停止遍历该产生式，进入下一个产生式；
6. 若已经到达产生式的最右部的终结符，则将$加入左部终结符的First集中，若不相等，则返回结束。

### 算法流程图

求取First集的算法流程图如图6.2所示。

图6.2 First集算法流程图



### 算法具体实现

求解First集的具体代码实现如图6.3所示。

图6.3 First集算法具体代码实现

## 求取Follow集

### 算法原理

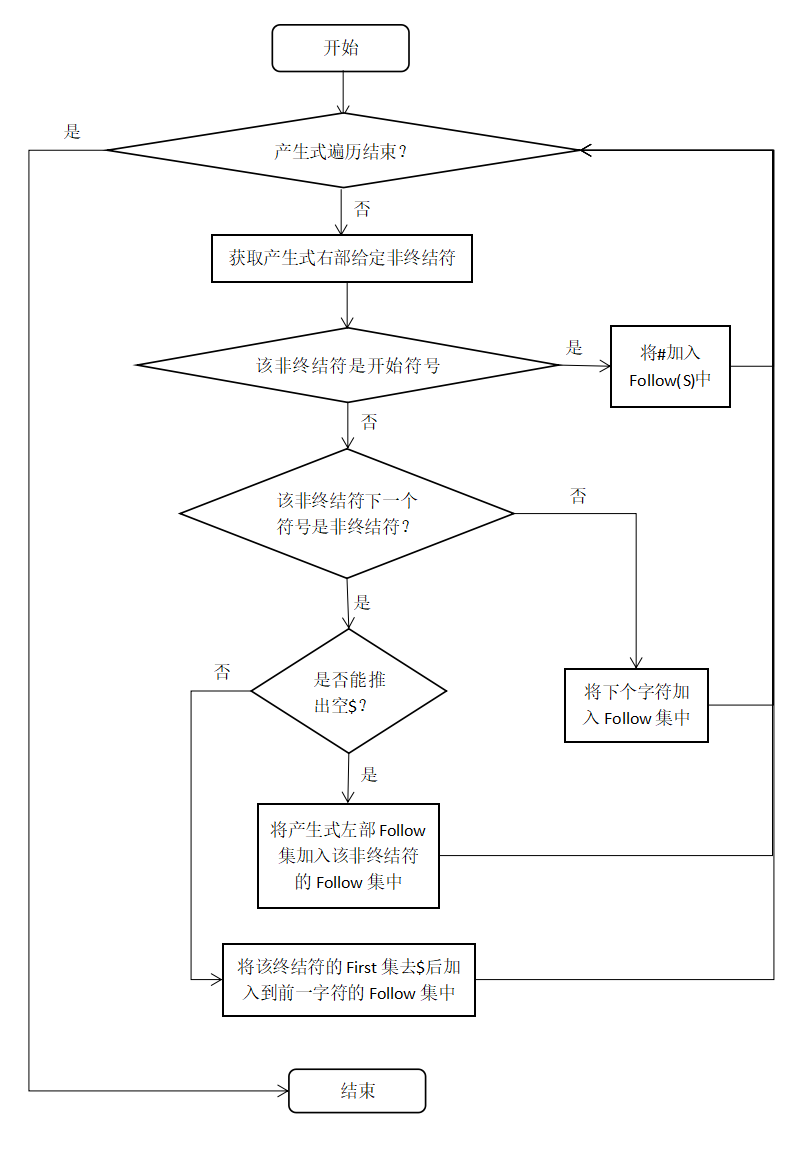
求取文法中非终结符的Follow集算法原理如下：

1. 设S 为文法的开始符号，则{#}加入Follow(S)中；
2. 若A->aBb是一个产生式，则将First(b)的非空元素加入Follow(B)中；
3. 若A->aB是一个产生式，或若A->aBb是一个产生式且b=>ε，则把Follow(A)加入Follow(B)中。
4. 反复使用上述步骤（2）、（3）直到每个非终结符的Follow集不再增大为止。

### 算法流程图

求取Follow集的算法流程图如图6.4所示。

图6.4 Follow集算法流程图



### 算法具体实现

求解Follow集的具体代码实现如图6.5所示。

图6.5 Follow集算法具体代码实现



## 求解Select集

### 算法原理

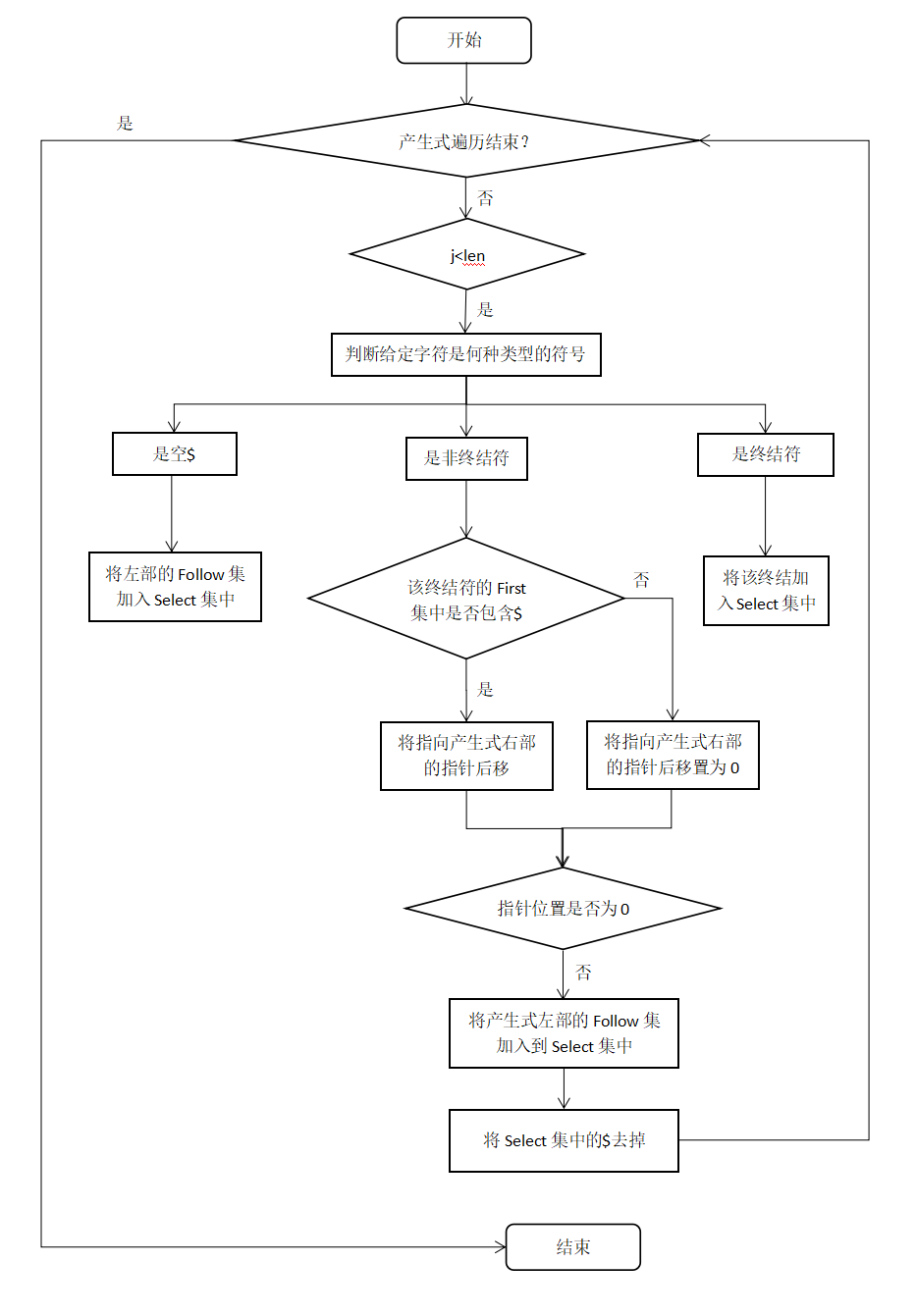
求取文法中所有产生式的Select集的算法如下：

1. 对所有产生式而言，若其右部第一个字符是$，则将左部的Follow集加入到该产生式的Select集中；
2. 若其右部字符串中第一个字符是终结符，则将该终结符加入到该产生式的Select集中；
3. 若其右部字符串是非终结符，则将其First集加入到该产生式的Select集中；
4. 若判断的字符串的First集中存在$,则将产生式左部Follow集并入到该产生式的Select集中；
5. 最后去掉Select集中包含的空串$。

### 算法流程图

求取Select集的算法流程图如图6.6所示。

图6.6 Select集算法流程图



### 算法具体实现

求解Select集的具体代码实现如图6.7所示。

图6.7 Select集算法具体代码实现



## LL(1)文法判定

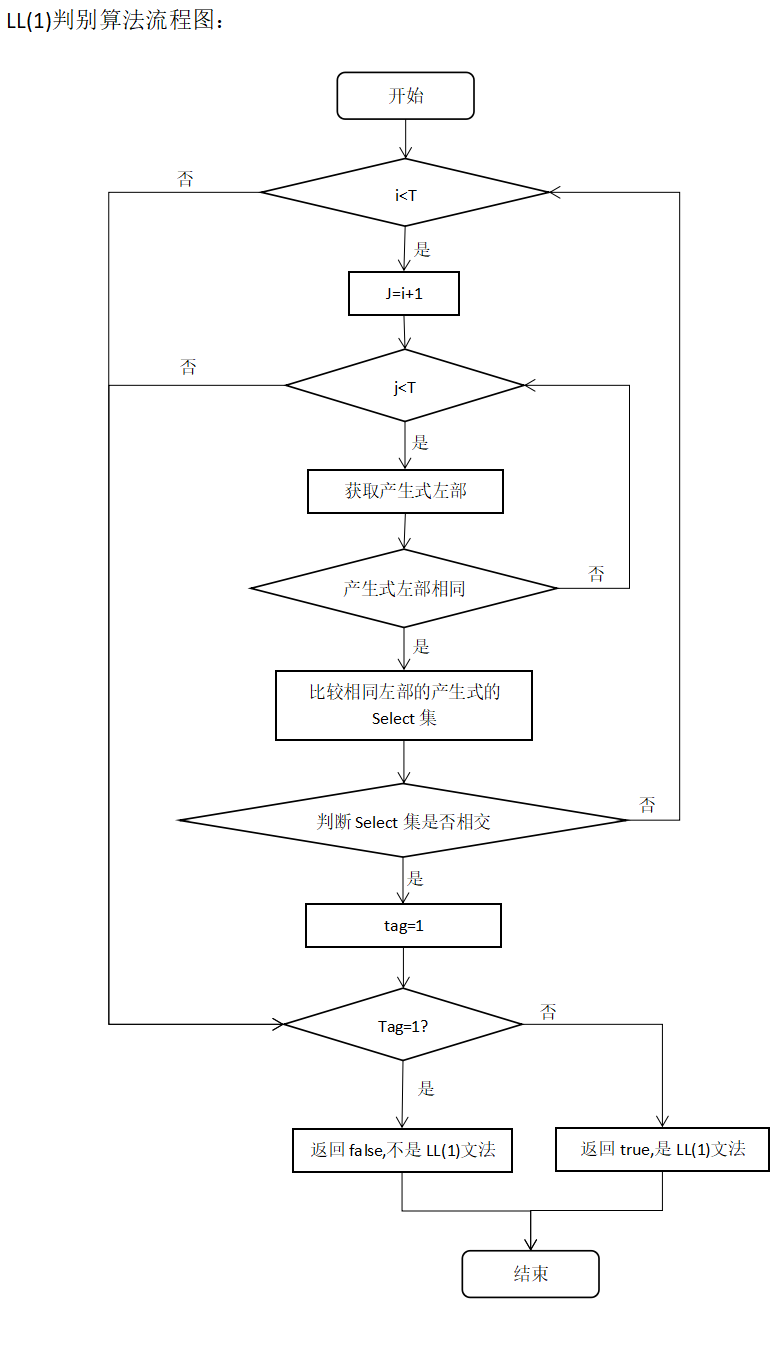
### 算法原理

LL(1)文法的判定可以通过判别LL(1)文法的充分必要条件来实现，即对每个非终结符的两个不同的产生式，其求取的Select集合两不相交，即它们的交集为空集。

### 算法流程图

LL(1)文法判别的算法流程图如图6.8所示。

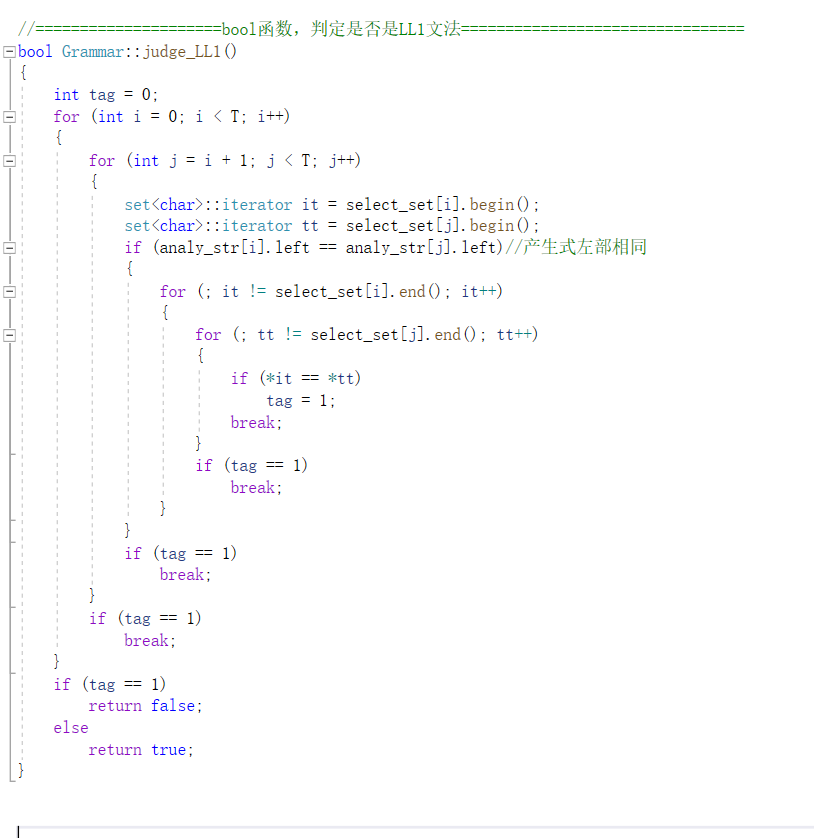
图6.8 LL(1)文法判别算法流程图



### 算法具体实现

LL(1)文法判别的具体代码实现如图6.9所示。

图6.9 LL(1)文法判别算法具体代码实现



## LL(1)预测分析表填写

### 算法原理

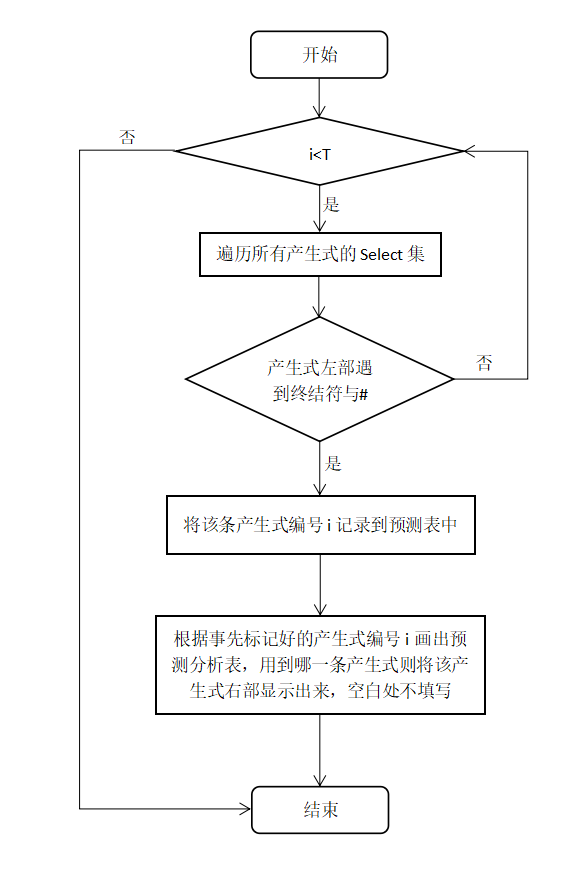
LL(1)预测分析表构造算法原理如下：

1. 遍历所有产生式的Select集，若产生式左部遇到终极符与#，则在预测分析表中标记上所用的产生式编号i；
2. 根据事先标记好了的产生式编号i画预测分析表，用到哪一个产生式就将该产生式的右部在此处显示出来，空白处则不填写。

### 算法流程图

LL(1)预测分析表填写的算法流程图如图6.10所示。

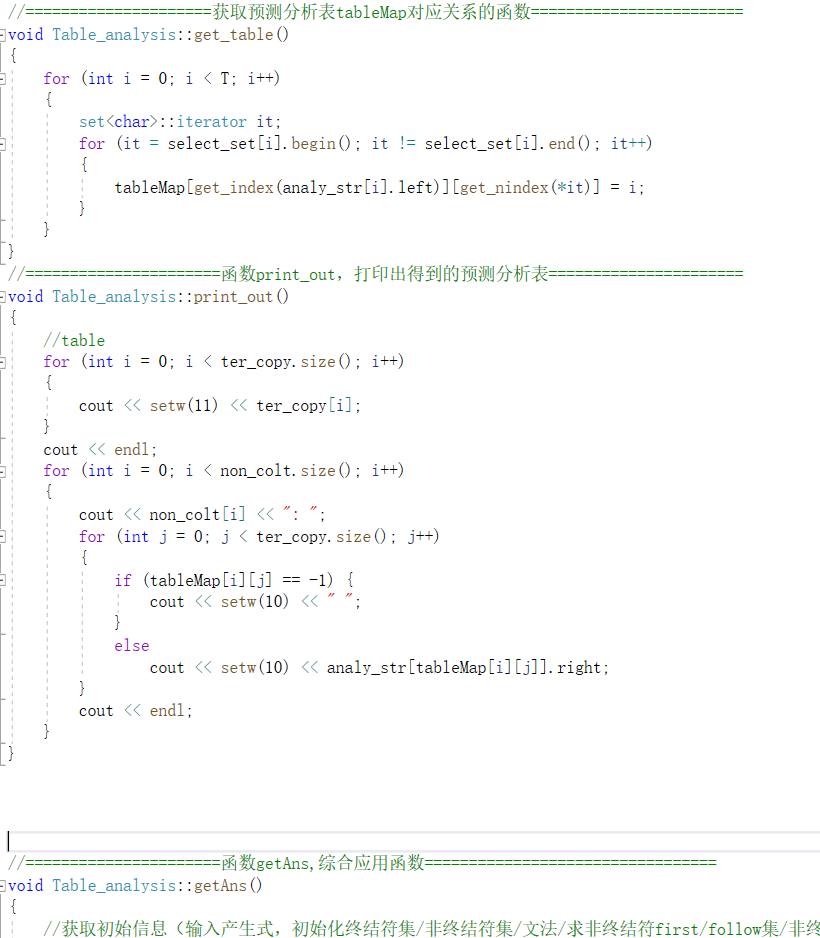
图6.10 预测分析表填写算法流程图



### 算法具体实现

LL(1)预测分析表填写算法的具体代码实现如图6.11所示。

图6.11 LL(1)预测分析表填写算法具体代码实现



## 运行结果与分析

### 测试用例

1. 文法测试用例1 (非LL(1)文法)：

S->CA

A->(C)

A->a

C->FB

B->A

B->$

F->i

1. 文法测试用例2 (LL(1)文法)：

E->TK

K->+TK

K->$

T->FN

N->FN

N->$

F->(E)

F->i

### 运行结果截图

输入两个文法测试用例进行测试，文法测试用例1所得运行结果如图6.12所示，文法测试用例2所得运行结果如图6.13所示。

图6.12 运行结果截图1

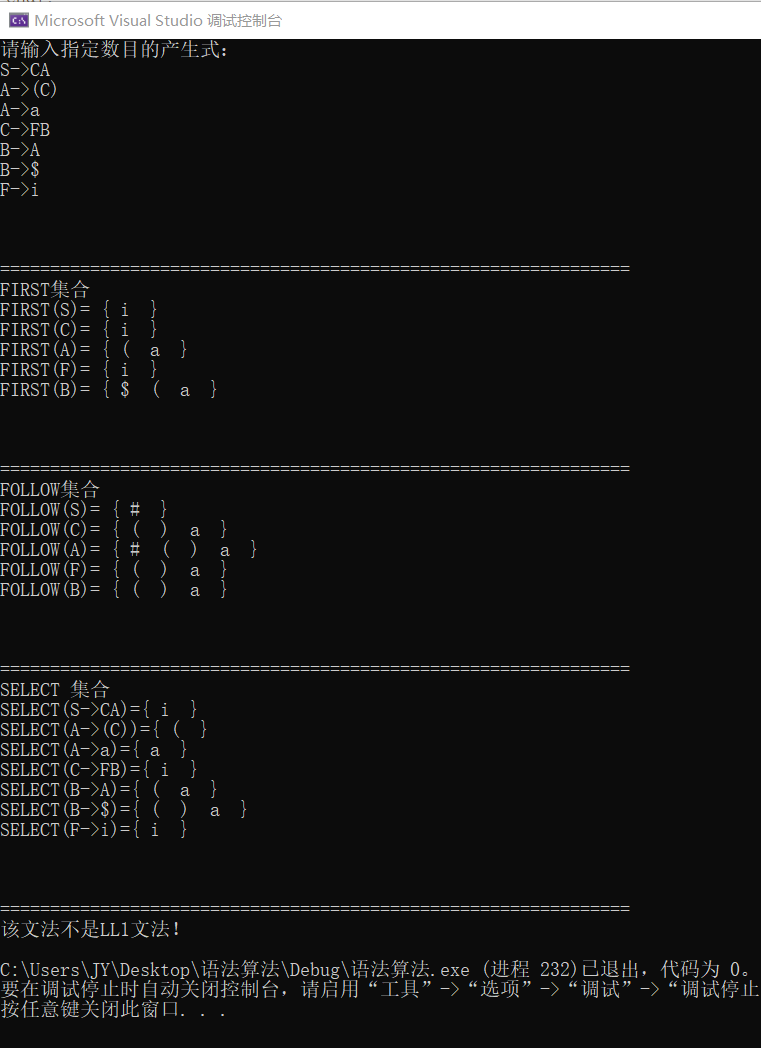
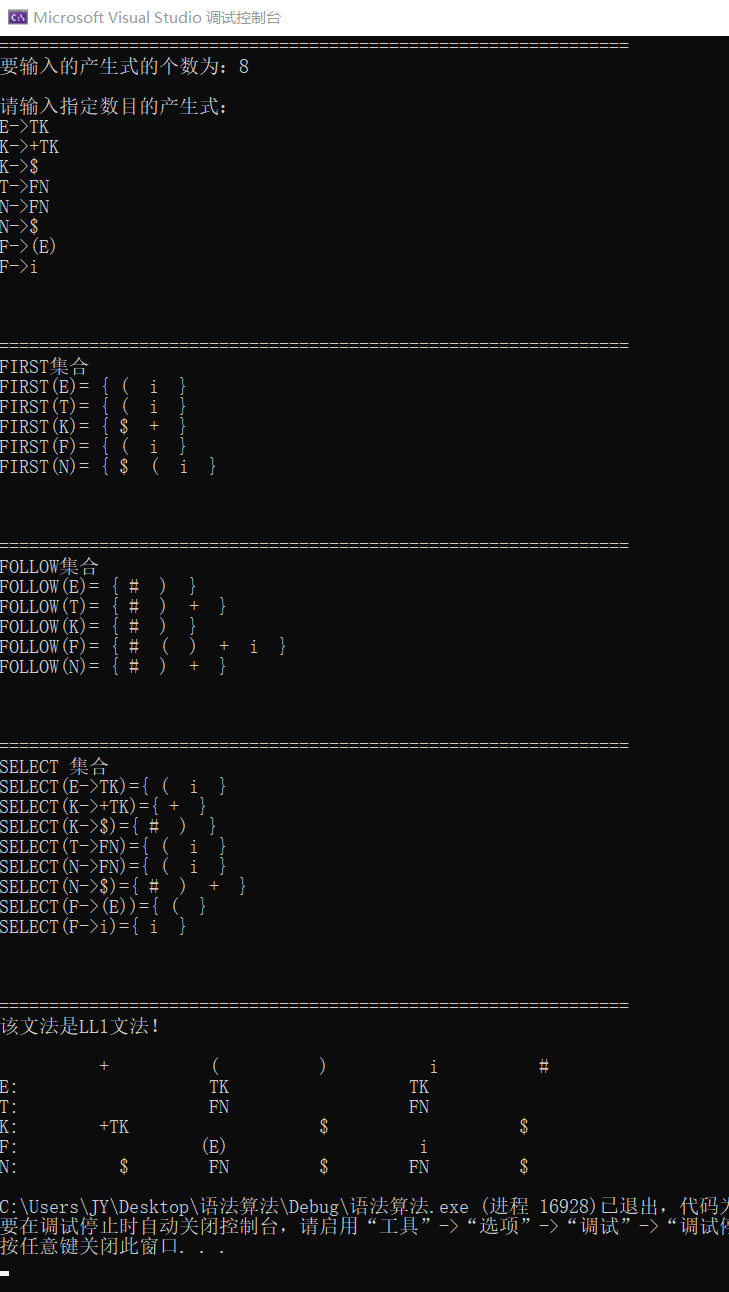


图6.13 运行结果截图2



### 运行结果分析

经过测试，能得到正确的运行结果。文法测试用例1中的产生式是不满足LL(1)文法的，而文法测试用例2中的产生式是满足LL(1)文法的，图6.12中的运行结果对应测试用例1，图6.13中的运行结果对应测试用例2。在测试时，先将要输入的产生式的个数输入到屏幕中，接下来输入指定个数的产生式，然后将求得的First集、Follow集和Select集进行输出，能够判断文法是否是LL(1)文法，如果是LL(1)文法则将其LL(1)预测分析表进行输出，否则程序直接结束。

从图6.12中的运行结果中可以看出，该程序能够正确输出非LL(1)文法的First集、Follow集、Select集并能正确的进行判断文法是否是LL(1)文法。文法测试用例1的产生式B->A、B->$具有相同的产生式左部，而他们的Select的交集为{ (，a }，不为空集，所以不是LL(1)文法，输出判断结果，程序直接结束，与预期结果一致。

从图6.13中的运行结果中可以看出，该程序能够正确输出LL(1)文法的First集、Follow集、Select集并能正确的进行判断文法是否是LL(1)文法。在文法测试用例2中，任何具有相同左部的产生式的Select集的交集均为空，所以该文法是LL(1)文法，输出判断结果并输出文法的LL(1)预测分析表，与预期结果一致。

# 语义分析程序设计及实现

7.1四元式结构体定义

struct formula //四元式结构体

{

char opr[20]; //操作码

char op1[20]; //操作数

char op2[20]; //操作数

char results[20]; //结果

int result;

};

其中opr数组储存操作码,op1和op2数组储存操作数,并用results数组存储结果。

7.2主要函数定义

（1）int Entry(char\* buf)函数是将buf加入符号表中的，具体操作是先查询符号表里面是否存在buf，如果存在就返回它的下标；如果不存在就加入到符号表的结尾，并返回此处的下标

（2）函数void GEN(string op,string arg1,string arg2,string result)

每调用一次，就根据代入的参数生成四元式，包括运算符、两个参数、跳转出口或者运算结果。

（3）函数void Backpatch (int n)

函数是用四元式序号t回填以p为首的链， 将链中每个四元式的Result域改写为t的值。

1. int NewTemp()函数是申请临时工作单元
2. void act(stack\* st, int num)函数是执行归约产生式对应的语义动作

7.3运行结果与分析

程序按照递归下降分析的思想开始对目标源程序进行语义分析，检查目标源程序是否以begin开头，若有则继续扫描，根据得到单词的类别码进行分析，例如num=5，即为IF，最终会生成四元式，扫描到end单词时，若后面紧跟着的单词为’.’，则认为源程序已经扫描完毕，程序结束执行并退出。

# 总结

1800301003莫敏华：

本次课程设计我主要负责的是语法分析模块当中的算法部分，即实现求解First集、Follow集合算法，判定文法是否为LL（1）文法的算法及LL（1）分析表填写算法。在进行编译原理理论学习的时候，我们已经学习过如何求解产生式的First集、Follow集等相关内容,不过经过了一段时间，对这些内容也已经忘记了一部分，所以一开始我选择先回归课本，经过复习能够手动计算出First集、Follow集、Select集。

在明确了求解的具体算法流程后，就开始进行具体的编码实现。虽然能够在纸上进行求解出First集、Follow集，但是到了实际写代码还是会有一定的难度，而且我编写代码的能力较弱，所以花了很多时间的时间去学习一些博客上的相关内容，去网上找了很多参考资料并借鉴了一些别人设计的思路，最后一点一点的能够实现求解First集、Follow集和Select集的算法，并通过select集判断文法是否是LL(1)文法。

通过本次的课程设计，我深切的认识到，理论和实践相结合的重要性，仅学习理论内容是远远不够的，在进行分析的过程中，手动求解的时候都知道求解的思路并且都能成功得到正确的结果，但是在实际操作过程中，往往就会出现很多问题，不知道这些求解过程的算法具体是如何应用到代码当中的，这就需要我们不断去查找资料进行学习，向老师同学请教。这次课程设计我学习了很多，在完成任务的过程中，对不同的算法有了更加深刻的认识和理解，能够将所学知识运用到实践当中。同时，也加强了我代码编写的能力，并且在小组合作的过程中更加明白了团队合作的重要性，与小组成员之间进行交流也提高了自己沟通交流的能力。

1800301338于彬磊：

在这次编译原理设计中，我主要负责的部分是语法分析的主程序编写。在这次项目中我了解到了项目细化分化，通过前者词法分析留下的文件对应相应的变换关系，变换表，对程序进行语法分析和检错。我也理解到理论与实际操作是不相同的，实际操作中需要引用许多抽象，与理论全抽象不同。同时也遇到了对于action,goto表如何表达的困惑，但最终都得到了解决。同时本次课程设计也加深了我对编译过程的理解。编译的源程序与目标程序中间变化过程的状态。加深了程序编写过程中的要点。更能理解理论知识。收获颇丰。

1801000711贺越华：

在这次编译原理课程设计中，我主要负责的部分是NFA确定化和DFA的最小化，在上个学期的学习中，我学会了如何进行NFA确定化和DFA的最小化，但是却没有通过编程的方式来实现，在这次课设刚刚开始的阶段，我还搞不清这部分算法和词法分析的关系，在老师的帮助下，终于知道要求是什么。在网上看了很多人的博客，终于有了如何写的思路，在几天都编程和调试过程后，最后实现了两个主要的算法。但是确没有规定状态的初始状态，这个程序就没有普适性了。但是通过这次小组合作实现整个编译器的过程，加深了我对理论知识的理解。同时也提升了自己团队合作的能力，由于自己的组织能力不够强，导致在课设前期任务分配不清，课设进度缓慢。在以后都工作和学习中也要不断提高自己的学习能力以及沟通能力。

1801000715刘炫均：

经过一个多月的编译原理课程设计，我们组在王老师的指导下，顺利完成该课程设计。通过该课程设计，收获颇多。我通过该课程设计，知道了什么是编译程序，编译程序工作的基本过程及其各阶段的基本任务，掌握了编译程序的词法分析的知识，明白了词法分析在一个编译程序所处的位置，词法是基础，完成好词法分析的工作才能进行下一步语法语义分析的工作，对课本上的知识有了更深的理解，对实验原理有更深的理解。通过在计算机上实现，对该词法分析在实践中的应用有深刻的理解。激发了我对学习的积极性。这次课设把学过的编译原理的知识重新学习一遍强化一遍，把课堂上学的知识通过自己设计的程序表示出来，对理论知识的理解更加透彻。

1801000728张家铭：

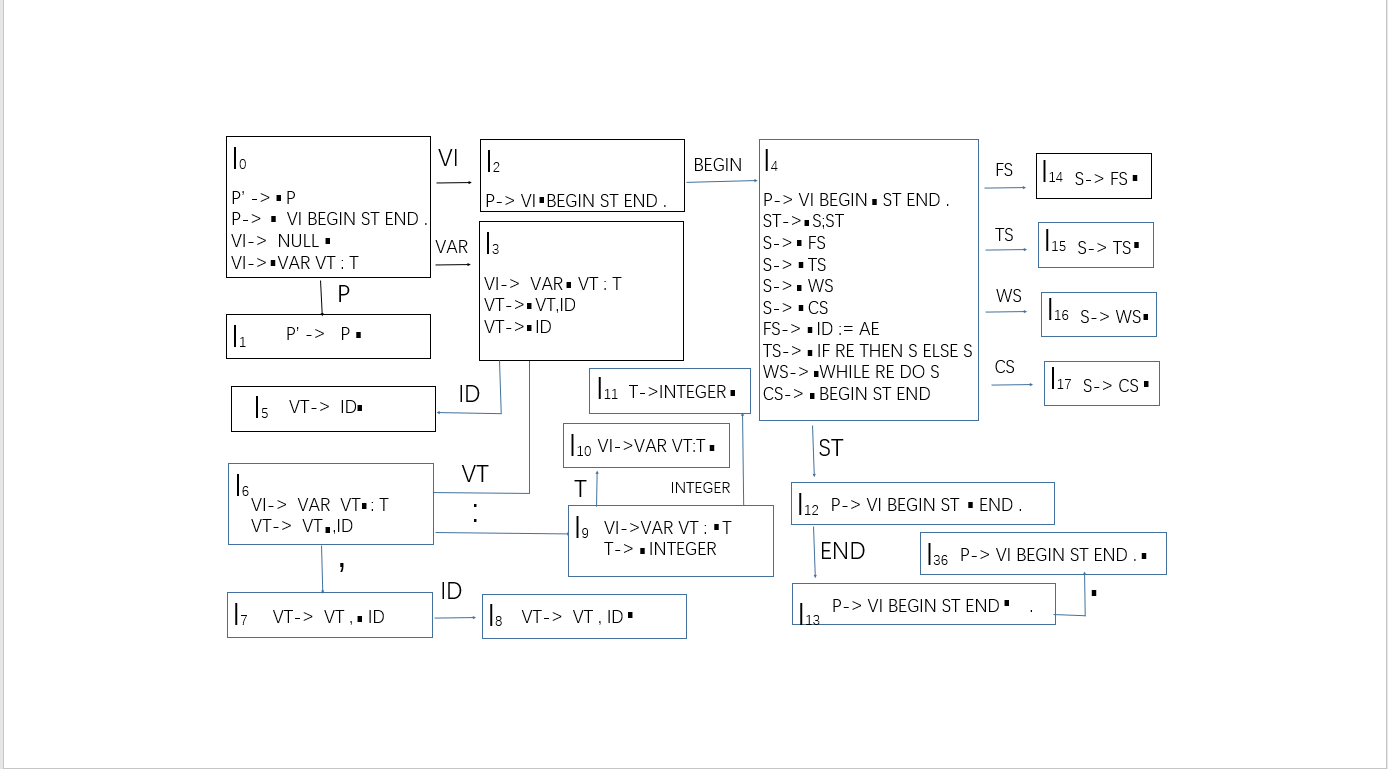
在本次课设中，我负责的是语义分析，它的任务是对结构上正确的源程序进行上下文有关性质的检查，并为代码生成阶段收集类型信息，以便于生成诸如四元式之类的中间代码。通过本次的编译原理的课设,我大大地加深了对编译原理这门课程的理解,同时我也明白了理论与实践结合的重要性,只有理论结合实践才能够真正地去参与到实际的问题中,提高独立思考和实际动手的能力。另外，我也明白了团队合作的重要性，在本次课程设计的实现过程中我们组遇到了许多问题，可以说是困难重重。在此，我要真诚地感谢老师以及我们组的同学。在每个人的共同努力下，这些问题都一一得到了解决，最后圆满的完成了本次的课程设计 。

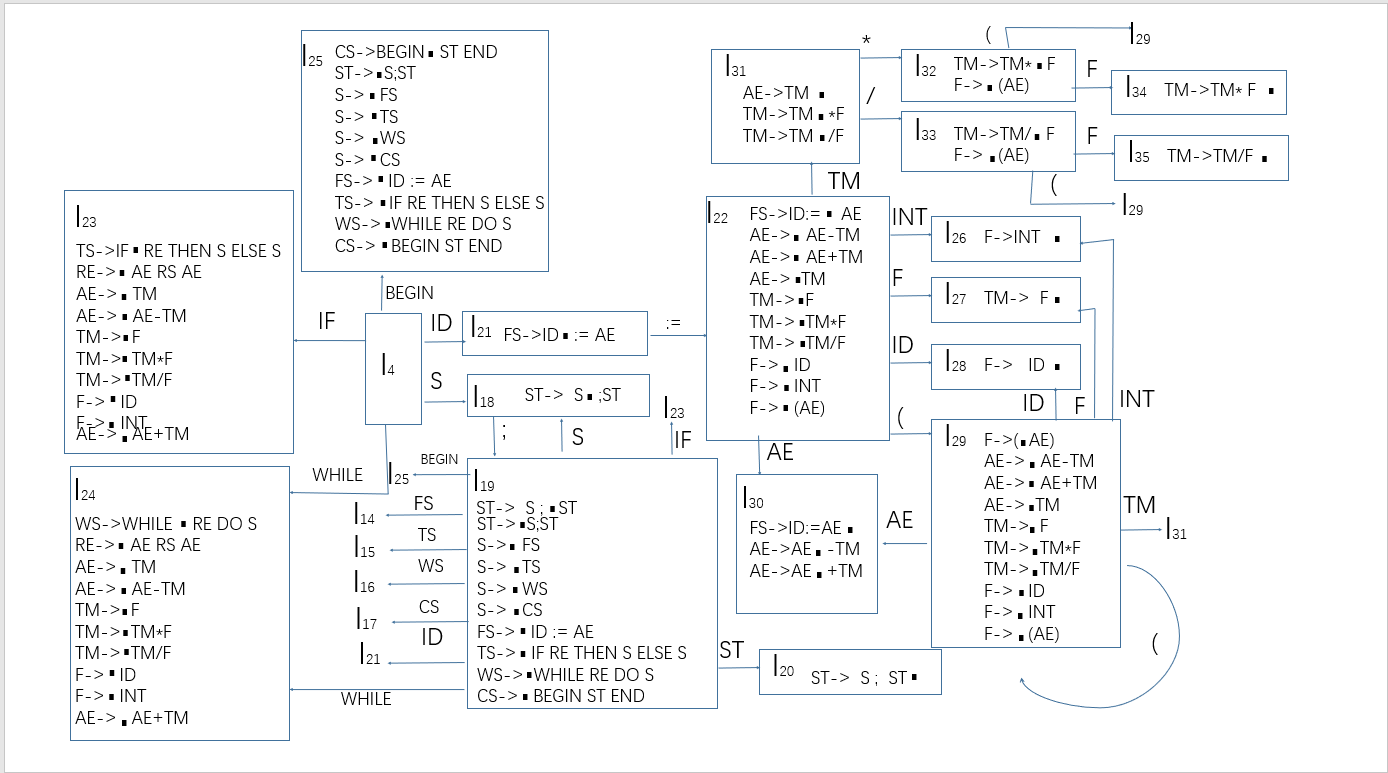
# 参考文献

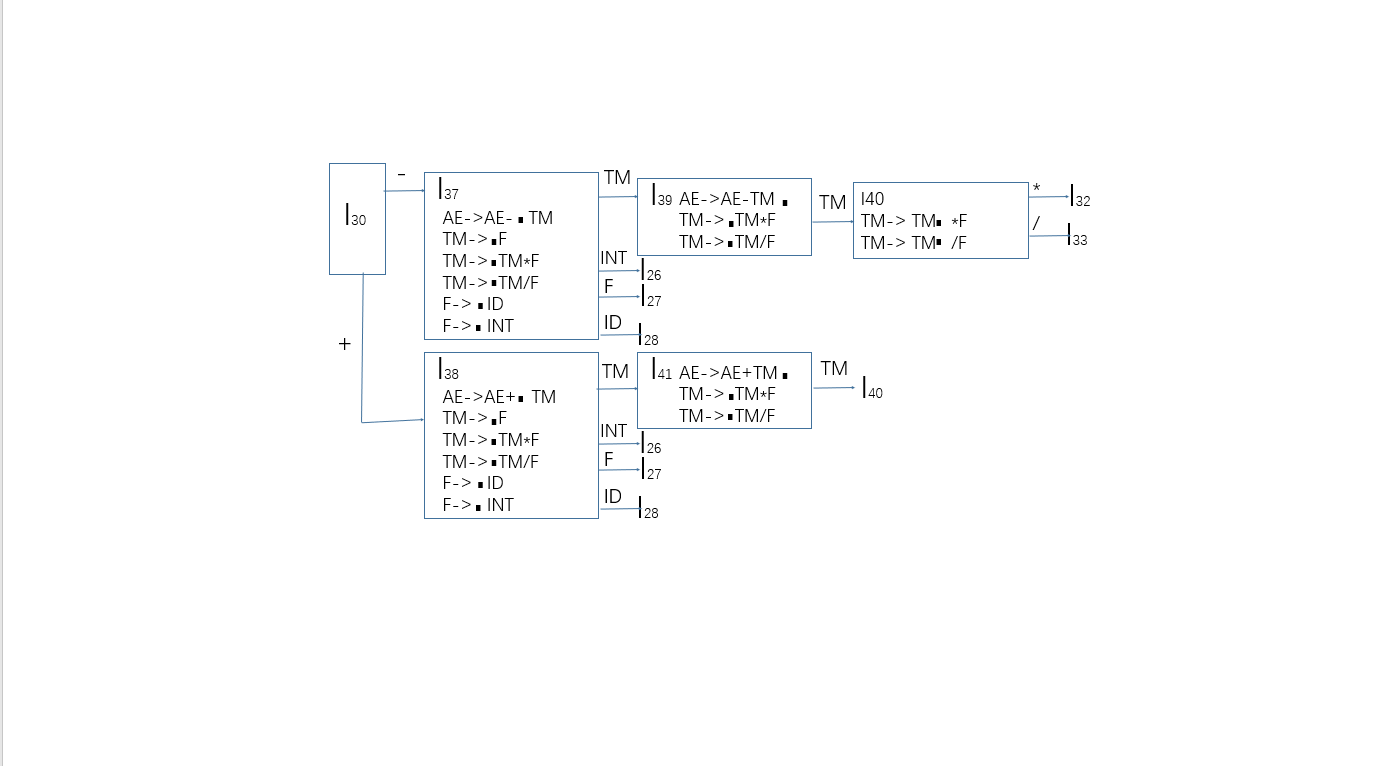
1. 王生原，董渊，张素琴，等. 编译原理（第3版）[M].北京：清华大学出版社，2015.
2. 蒋立源,康慕宁. 编译原理（第3版）[M].西安：西北工业大学出版社，2005.
3. 陈火旺. 程序设计语言编译原理（第3版）[M].北京：国防工业出版社，2000.
4. 杜淑敏，王永宁. 编译程序设计原理[M].北京：北京大学出版社，1986.
5. 金成植. 编译程序构造原理和实现技术[M].北京：高等教育出版社，2002.

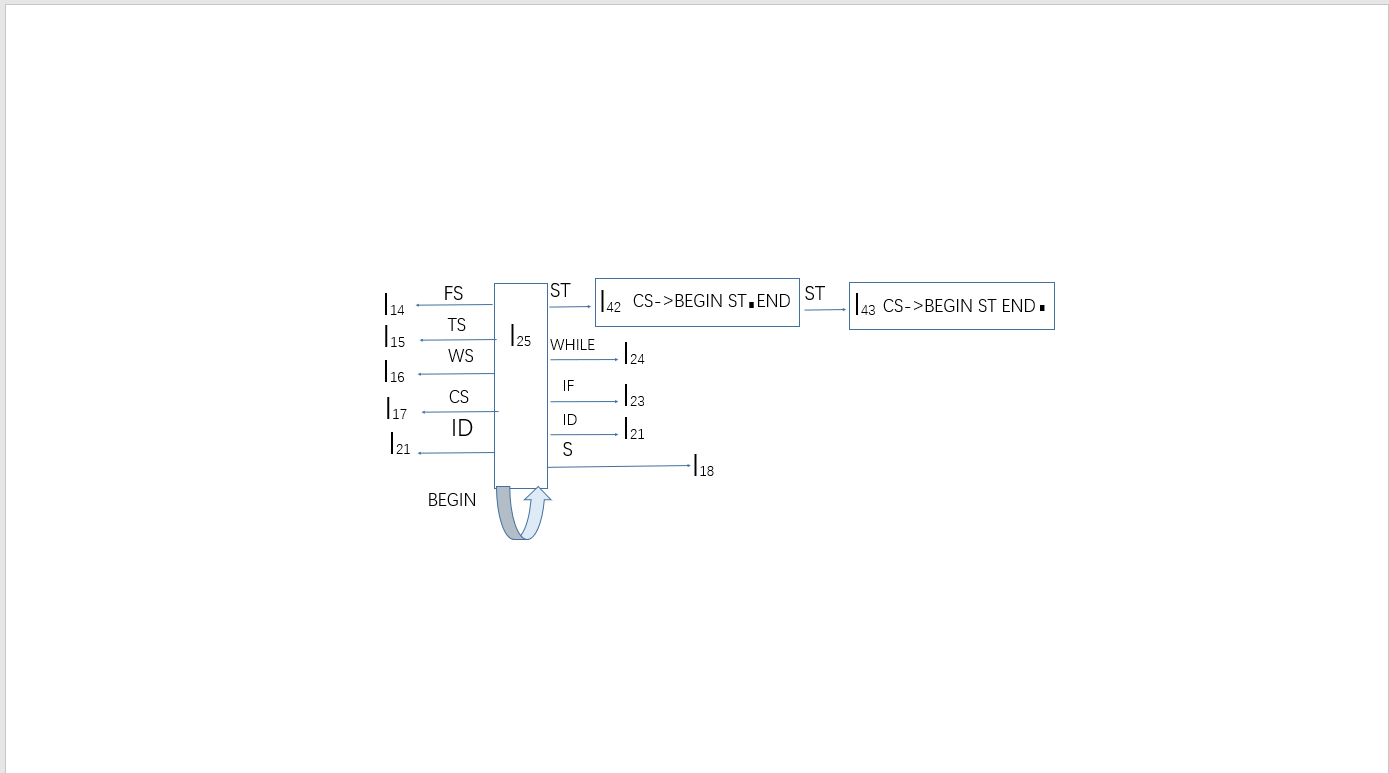
附录一

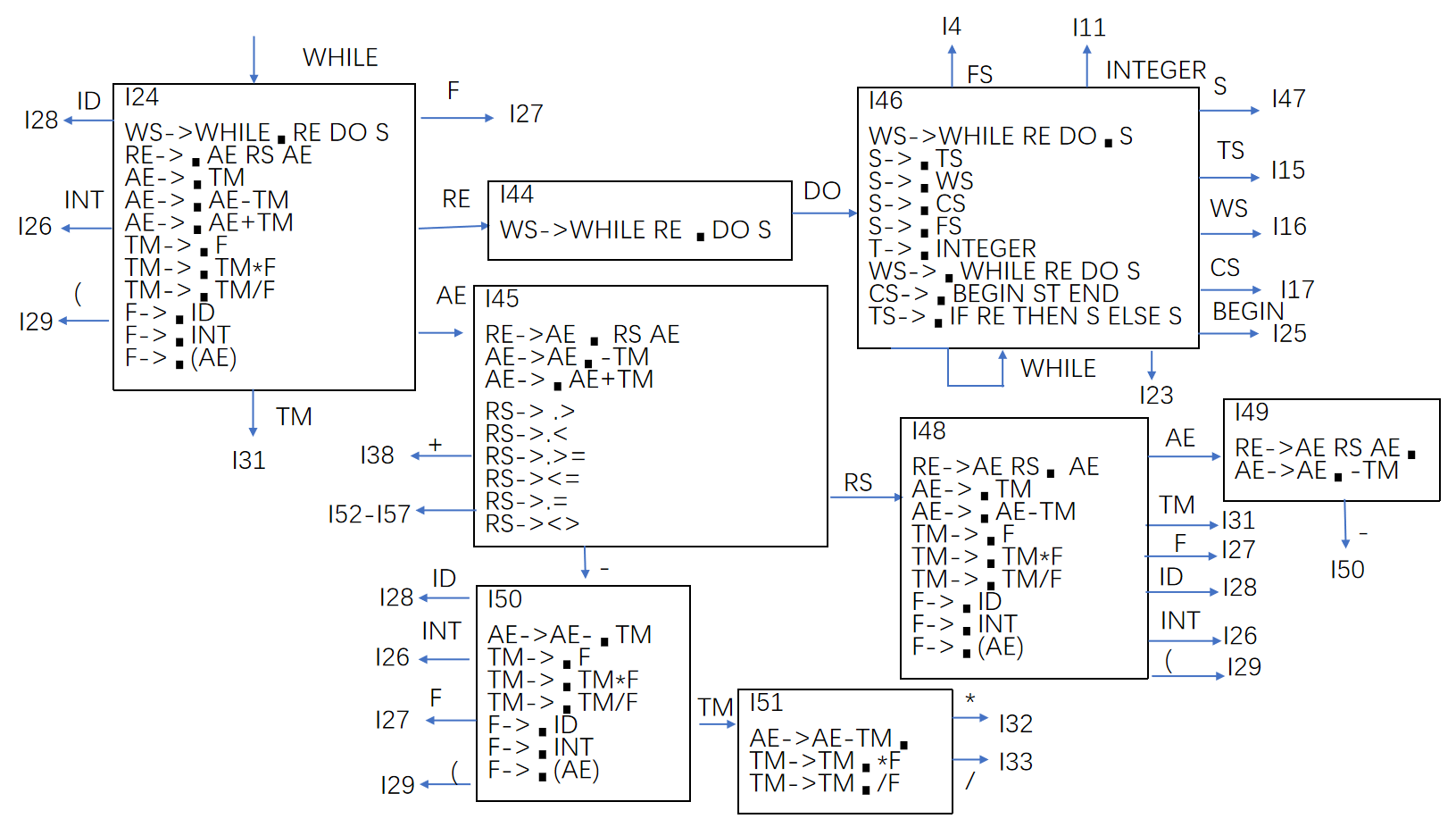
DFA构造表

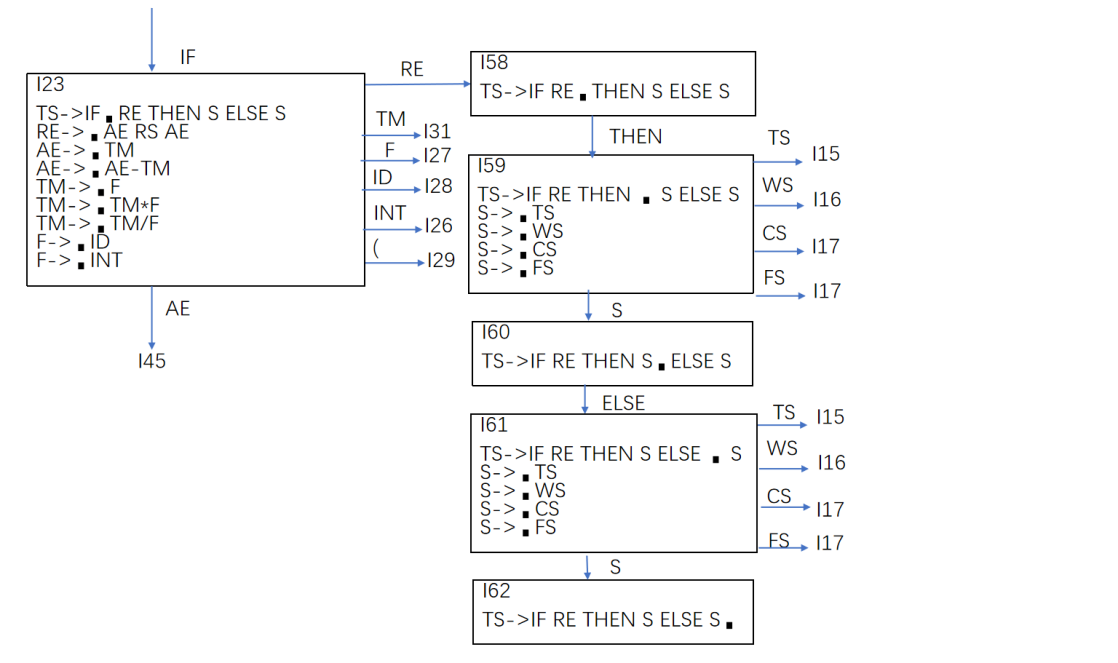












附录二

SLR文法分析表

