编号：

编译原理课程设计报告

题 目： 一个Simple-PASCAL编译器

的设计与实现

院（系）：

专 业：

团队成员：

指导老师：

职 称：

2020年6月20日

摘 要

编译器就是语言翻译器，它一般将高级语言翻译成更低级的语言，如 GCC 可将 C/C++ 语言翻译成可执行机器语言，同样的，PASCAL语言也是一种计算机通用的高级[语言](http://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80)，是一门结构化编程语言。

本课程设计要求实现一个PASCAL语言小型编译器的开发，因为PASCAL语言相对其他语言，其严格的结构化形式使该语言的查错能力强，十分适合作为本门课程设计的目标编译语言。在这次课程设计中，我们将按照编译器通常对程序进行编译的步骤进行开发，从词法分析到语法分析再到语义分析，最终实现简单的编译器。

本课程设计报告首先给出了关于做PASCAL语言小型编译器的任务和实现目标，阐述小组成员的分工和安排，通过分别对词法、语法和语义实现的过程进行分析，规定相关的单词和产生式编号，画出相应的DFA构造表和SLR流程图，进而给出实现编译操作的算法代码，最后给出对应的结果测试和分析情况。

本门课程设计中，我们将会使用C语言编写一套符合PASCAL语言语法规则，使其能够将简单的PASCAL语言编译。

关键词：编译原理；编译器；词法分析；语法分析；语义分析

**目 录**

[1 概述 1](#_Toc30696)

[2 课程设计任务及要求 2](#_Toc14163)

[3 词法分析程序设计及实现 2](#_Toc24071)

[3.1 二元式 3](#_Toc803)

[3.2 单词分割 3](#_Toc30336)

[3.3 单词的分类与编码 5](#_Toc15186)

[3.4 词法分析主要算法 5](#_Toc17632)

[4 语法分析程序设计及实现 8](#_Toc19623)

[4.1 SLR文法的构造 8](#_Toc17903)

[4.2 语法分析程序设计与实现 11](#_Toc22158)

[4.3 主要模块的算法 11](#_Toc25978)

[5 语义分析程序设计及实现 17](#_Toc5069)

[5.1 语义分析设计方法 17](#_Toc24384)

[5.2 语义分析程序设计 18](#_Toc11487)

[5.3 主要模块的算法 19](#_Toc12700)

[6 系统结果及分析 23](#_Toc22406)

[6.1 测试案例 23](#_Toc29831)

[6.2 词法分析结果 24](#_Toc22265)

[6.3 语法分析结果 25](#_Toc4654)

[6.4 语义分析结果 26](#_Toc25103)

[7 总结 27](#_Toc422)

[参考文献 29](#_Toc8137)

[附录一 30](#_Toc23247)

[附录二 32](#_Toc19344)

1 概述

为了提高效率，有效分工。我们小组将编译器实现的过程模块化，分别是词法分析、语法分析、语义分析。编译程序的输出结果包括词法分析后的二元式序列、变量名表；语法分析后的状态栈分析过程以及四元式序列。为了能够系统化、结构化进行PASCAL语言编译器的设计，我们使用C语言作为编译器的编程语言。

各阶段情况如下：

1. 词法分析

此阶段主要是从左到右一个字符一个字符地读入源程序（PASCAL语言），对构成源程序的字符流进行扫描然后根据构词规则识别单词。该模块会输出二元式流。

1. 语法分析

此阶段的主要是构建抽象语法树，处理词法分析得到的二元式流，检查是否存在语法错误。这里，我们采用SLR分析来构建该语法树。该模块会输出相对完整的状态栈分析过程。

1. 语义分析

此阶段编译器会遍历上一步生成的抽象语法树，然后为每个节点生成四元式序列。

# 2 课程设计任务及要求

2.1 词法分析

利用给出的类PASCAL文法，以Flex、bison及C语言等工具编写的可被正确编译运行的源程序，将写好的源程序作为输入，进行词法分析并给出分析结果。

要求：

（1）有正确的单词流输出（类型码、坐标、单词的值）

（2）能够检查单词的拼写错误，并报错

（3）能够识别注释，并去掉注释

（4）整数能够处理越界

（5）标识符识别前8位

2.2 语法分析

利用词法分析的结果或者调用词法分析模块进行语法分析程序设计。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，检查是否存在语法错误。

要求：

（1）能够检查语法错误

（2）能够报告语法错误

（3）能够语法错误恢复

2.3 语义分析

针对分析法中所使用的文法，为其设计翻译方案，并利用该翻译方案，对所给程序段，进行分析，生成中间语言。中间语言采用通用的四元式格式。

要求:

（1）能够对所给程序段，进行分析。

（2）能够实现回填

# 3 词法分析程序设计及实现

对于词法分析，其主要的流程包括了对输入字符串流进行分割，将输入文件分割成为正确的单词序列，随后判断单词的类型，生成二元式链表，最后对词法分析中发生的错误进行提示，其流程图如图3-1所示。

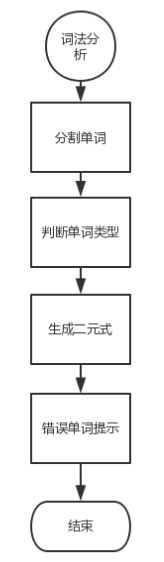


图3-1 词法分析流程图

## 3.1 二元式

二元式是用来存储每一个词类型以及参数等的数据结构，贯穿整个编译器设计的数据对象，它由词法分析器生成，语法分析和语义分析都使用这个内容作参考。其结构体定义如下：

struct ds

{

char \*idstr; //字符串

int style; //类型

};

struct ds data[MAX];

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

3.2 单词分割

词法分割，即将文件输入的pascal语言程序分割成token串的序列。

由语法生成式，我们可以知道各种词法的结束符，如常数的结束符为非数字，英文单词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，其结束符可以是数字/字母/符号。

在设计分割的判断规则上，我们一般读第一个字符后再读第二个字符，依此根据结束符来判断输出的token的情况（如判断标识符和关键字），具体规则判断流程如图3-2 所示。

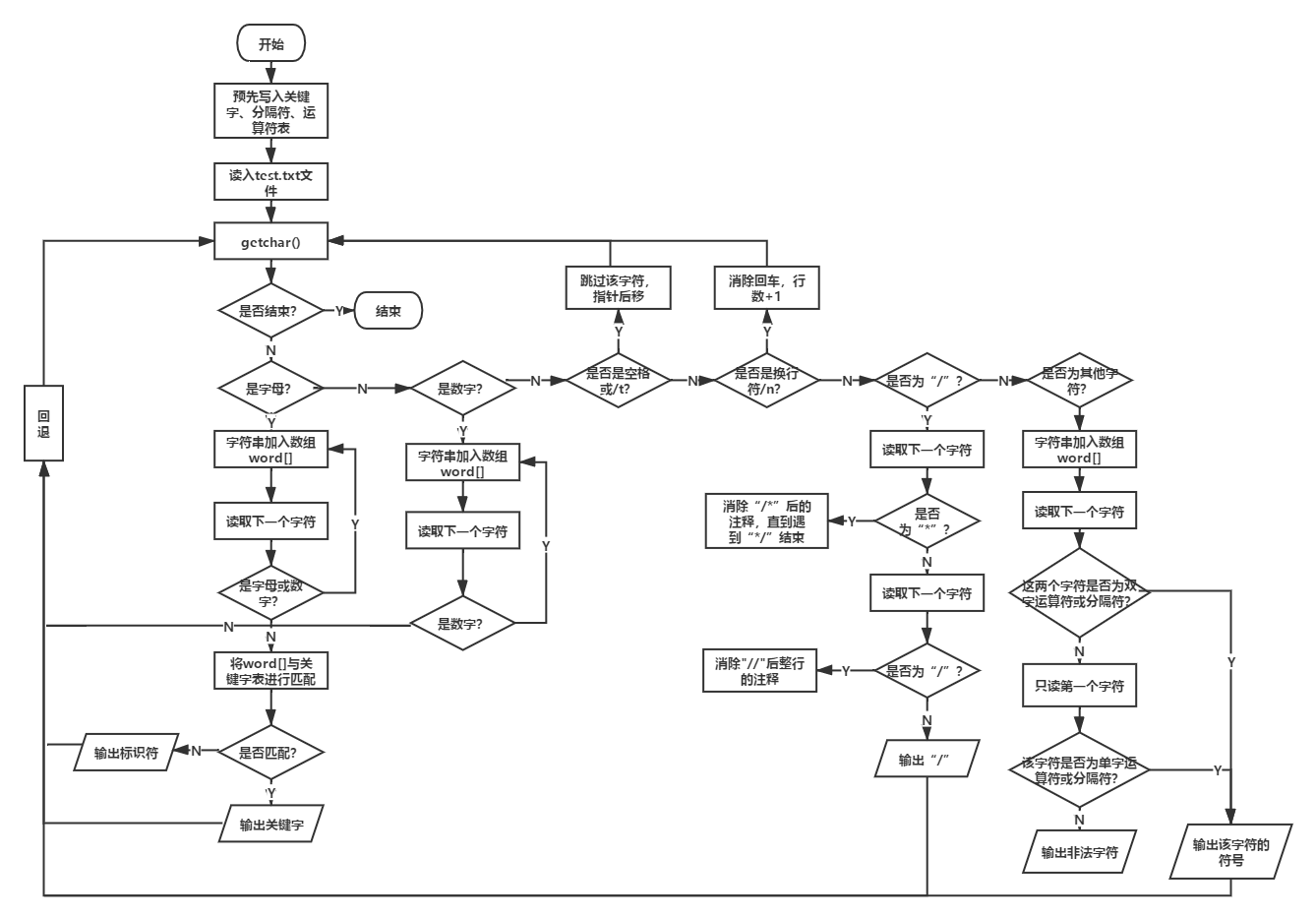


图3-2 词法分析流程图

3.2.1 初始化符号表

将保留字、分隔符、运算符所包含的符号分别预先写入keyword、separator、opration.txt文件中；创建标识符id.txt和整数符int.txt，和最终词法二元式输出的token.txt文件。

* + 1. 读入test.txt文件，扫描单词、分割、判断单词类别，写入token文件：

（1）取出首字母

①首字符以字母开头：取出连续的字母或数字，加入word[]数组，调用ch\_manage()函数先判断字符条件是否符合，再判断是关键字还是标识符；

②首字符以数字开头：该连续字符串为数字，调用int\_manage（），根据find（）函数command=2将数字字符添加入整数表；

③消除字符：分别需要消除空格和‘\t’、消除回车‘\n’、消除注释‘/ ’

其中，下一个字符不是‘\*’也不是‘/’：则‘/’表示是除号，类型号记为15；下一个字符是‘/’：则消除‘ // ’后面整行的注释；下一个字符是‘\* ’：消除‘/\* ’后面的注释，直到遇到‘\*’且后面是‘/’结束；

④其他字符：先读入双字符，find()判断是否为双字运算符或分隔符，若不是，读入单字符检测，find()判断是否为双字运算符或分隔符，是，同样写进token；不是，则为非法字符，调用error\_manage()函数报出出错符号和出错行；

（2）出错处理：字母字符串长度>8，数字字符串长度>5，符号值>MAXNUM，即报出出错符号和行数。

3.3 单词的分类与编码

对二元式产生的单词编码类型如表3-1所示：

表3-1 单词分类和编码表

|  |  |  |  |
| --- | --- | --- | --- |
| **单词** | **编码** | **单词** | **编码** |
| # | 0 | / | 15 |
| 标识符 | 1 | < | 16 |
| 正整数 | 2 | <= | 17 |
| BEGIN | 3 | > | 18 |
| END | 4 | >= | 19 |
| IF | 5 | <> | 20 |
| THEN | 6 | = | 21 |
| ELSE | 7 | ; | 22 |
| WHILE | 8 | . | 23 |
| DO | 9 | := | 24 |
| INTEGER | 10 | , | 25 |
| VAR | 11 | ( | 26 |
| + | 12 | ) | 27 |
| - | 13 | : | 28 |
| \* | 14 |  |  |

3.4 词法分析主要算法

3.4.1 识别单词及分类函数的算法

以字母开头为例：

if(((ch>='A')&&(ch<='Z'))) //以字母开头

{ while(((ch>='A')&&(ch<='Z'))||((ch>='0')&&(ch<='9')))

{

array[i++]=ch;

ch=fgetc(fpin);

}

word=(char \*)malloc((i+1)\*sizeof(char));

memcpy(word,array,i);

word[i]='\0';

ch\_manage(word); //处理字符串

if(wrong) //提示出错

{ errorno++;

printf("\nerror:out of range :");

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

printf("%c",word[k]);

printf(" ,line: %d", line);

}

wrong=0;

if(ch!=EOF)

fseek(fpin,-1L,1); //文件指针后退一个字节

}

3.4.2 查填符号表的算法

①如在ch\_mangae()函数中，根据查表函数find()先查关键字表，判断出该单词是否为关键字，令command=1（只查表）：

若是，返回表中序号，取result值为类号值（保留字都是result=3)， 写入token文件；若不是，返回0，调用find()函数查标识符表，若是，一样的取result，写入token文件；

若两者都不是，令command=2（要造表）：向表中添加标识符。

②用case()进行类别判断，并最终归类到二元式中，写入token文件。

int find(char \*buf,int type,int command)

{

int number=0;

FILE \*fp;

char c;

char temp[30];

int i=0;

switch(type)

{

case 1: fp=fopen("id.txt","r");break;

case 2: fp=fopen("int.txt","r");break;

case 3: fp=fopen("keyword.txt","r");break;

case 4: fp=fopen("operation.txt","r"); break;

case 5: fp=fopen("separator.txt","r");

}

c=fgetc(fp);

while(c!=EOF)

{

while(c!='\n')

{

temp[i++]=c;

c=fgetc(fp);

}

temp[i]='\0';

i=0;

number++;

if(strcmp(temp,buf)==0)

{

fclose(fp);

return(number); //若找到，返回在相应表中的序号

}

else

c=fgetc(fp);

}

if(command==1) //找不到，当command=1，返回0

{

fclose(fp);

return(0);

}

switch(type) //当command=2,向表中追加标识符

{

case 1: fp=fopen("id.txt","a");break;

case 2: fp=fopen("int.txt","a");break;

case 3: fp=fopen("keyword.txt","a");break;

case 4: fp=fopen("operation.txt","a");break;

case 5: fp=fopen("separator.txt","a");

}

fprintf(fp,"%s\n",buf);

fclose(fp);

return(0); //将字符串添加到表尾

}

# 4 语法分析程序设计及实现

## 4.1 SLR文法的构造

4.1.1 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。

4.1.2 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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **非终结符** | **单词化** | **编码** |  | **序号** | **单词化** | **编码** |
| <程序> | P | 29 | <算术表达式> | AE | 37 |
| <变量说明> | VI | 30 | <关系表达式> | RE | 38 |
| <变量表> | VT | 31 | <关系符> | RS | 39 |
| <语句表> | VT | 32 | <赋值语句> | FS | 40 |
| <语句> | S | 33 | <while语句> | WS | 41 |
| <类型> | T | 34 | <复合语句> | CS | 42 |
| <因式> | F | 35 | <条件语句> | TS | 43 |
| <项> | TM | 36 |  |  |  |

1. 将文法的产生式编码，如表4-2所示；

表4-2 产生式编码

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **编码** | **产生式** |  | **编码** | **产生式** |  | **编码** | **产生式** |
| ① | VI->NULL |  | TM->TM\*F |  | S->TS |
| ② | VT->ID |  | TM->TM/F |  | S->WS |
| ③ | T->INTEGER |  | F->(AE) |  | S->CS |
| ④ | VT->VT,ID |  | WS->WHILE RE DO S |  | RS-> < |
| ⑤ | ST->S |  | CS->BEGIN ST END |  | RS-> <= |
| ⑥ | F->ID |  | RE->AE RS AE |  | RS-> > |
|  | F->INT |  | TS->IF RE THEN S ELSE S |  | RS-> >= |
|  | TM->F |  | AE->AE-TM |  | RS-> = |
|  | AE->TM |  | P->VI BEGIN ST END . |  | RS-> <> |
|  | FS->ID := AE |  | VI->VAR VT : T |  | ST->S;ST |
|  | AE->AE+TM |  | S->FS |  |  |

（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

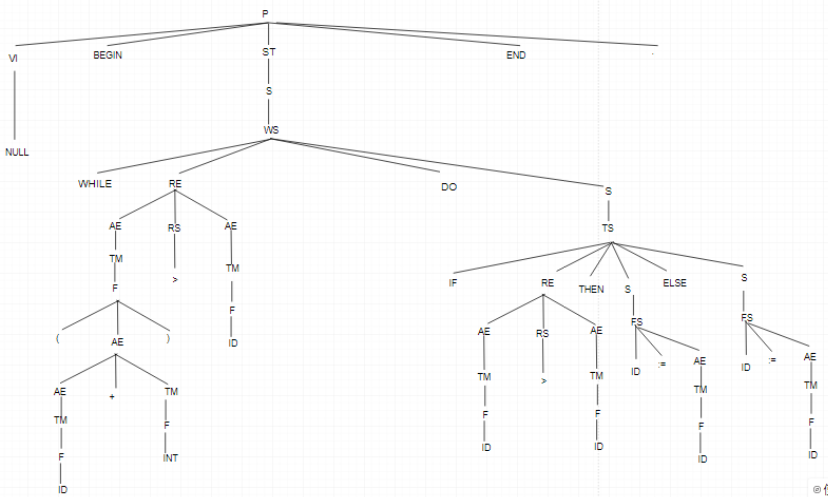
X:=Y+Z

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）。

4.2 语法分析程序设计与实现

4.2.1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

|  |  |
| --- | --- |
| **结构体/变量名称** | **作用** |
| struct stack | 与pop（）函数、push( )函数共同作用，模拟栈。通过对该栈的操作，实现状态栈、符号栈等信息的更新。 |
| struct inds input[MAXSIZE]; | 该结构体用来存放词法分析得到的结果(token.txt),包括单词和单词对应编码。 |
| struct tableproduction production[33] | 模拟产生式属性表，当action表的状态是规约(‘r’)时查询该表。 |
| struct tableaction ACTION[63][29] | 模拟SLR分析表的action表。 |
| int GOTO[48][15] | 模拟SLR分析表的GOTO表。 |
| char \*string[MAXSIZE] | 存储符号表 |
| int error | 记录当前错误个数 |

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

|  |  |
| --- | --- |
| **方法名称** | **作用** |
| push（） | 入栈 |
| pop（） | 弹栈 |
| Lrdriver（） | 模拟SLR分析器进行SLR分析并输出分析过程 |
| yufa() | 将词法分析得到的token表、写好的ACTION表、GOTO表、产生式表等表的信息读入对应的变量中；再调用Lrdriver（）方法实现语法分析 |

## 4.3 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”。

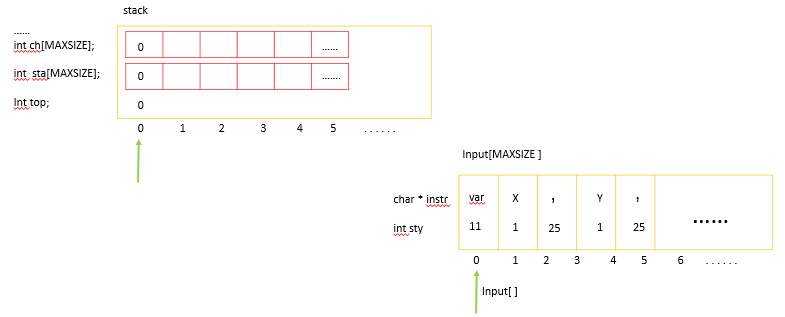
首先将单词存入input[ ] 数组，并初始化栈（加入#），如图4-2所示

图4-2 初始化栈

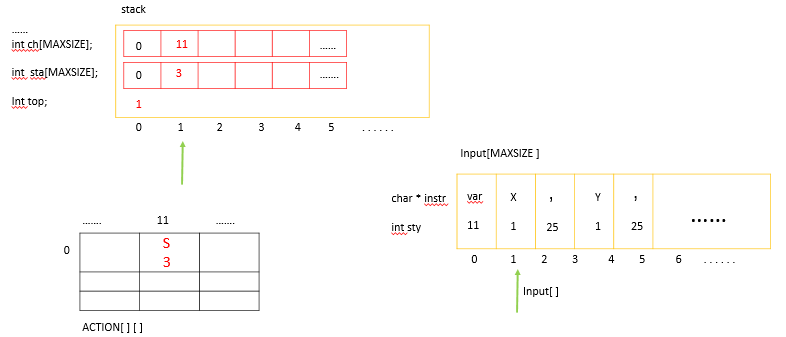
这时，根据栈stack的元素sta[0] 、和input[ 0] 的元素sty的值作为action表坐标查action 表，此时即为action [ 0] [ 11]。查表发现，action表该坐标下状态是“S”，故进行入栈操作，结果如图4-3所示。

图4-3 移进1

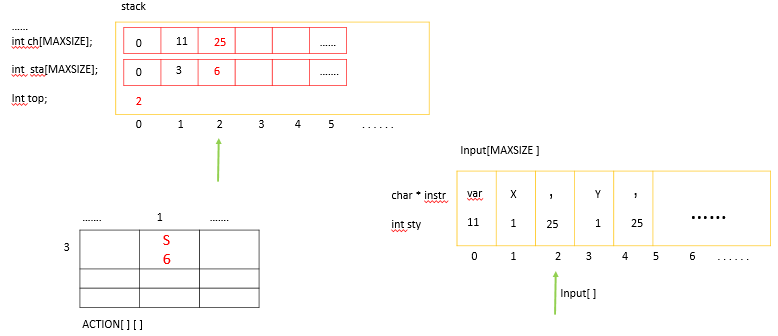
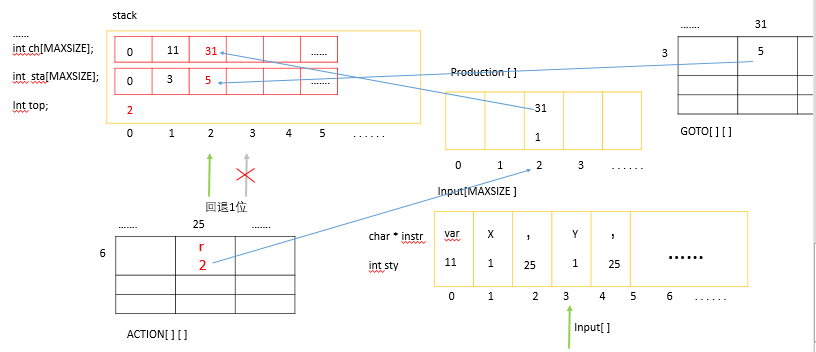
这时，栈中的状态栈就是03，符号栈就是011，栈顶元素为1。下一步对单词“，”进行操作，过程类似上一步，在此不再赘述，结果见图4-4。

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; //当前输入符号(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>=6)

printf("\t");

else

printf("\t\t");

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

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

if(l>=6)

printf("\t");

else

printf("\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）

{

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

intop++;

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

}

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

{

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

PL=production[ACTION[TS][IS].s].word\_number;

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

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);

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

}

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);

printf("\tAccepted!\n");

}

else

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

};

# 5 语义分析程序设计及实现

## 5.1 语义分析设计方法

### 5.1.1 S属性翻译方法

我们采用S属性翻译方法，语义动作被嵌入到产生式右部的适当位置，在推导过程中完成语义处理，中间代码用四元式表示，以下改进的属性翻译文法为将原产生式“拆分”成若干个“小”产生式，拆分符设在每个动作符的出现处，以保证动作符均出现在最右侧：

(1) 赋值语句：FS->ID := AE

动作解释：

{GEN(:=,$3.PLACE,0,$1.PALCE);}

意义及功能：将表达式AE求值，并赋于标识符ID。

（2）条件语句：TS->IF RE THEN S(1) ELSE S(2)

动作解释：

TS->IF RE THEN

{BackPatch(RE.TC,NXQ);}

S(1) ELSE

{T=NewTemp();T=Merge(NXQ,S(1).Chain);

GEN(j,0,0,0);

BackPatch(RE.FC,NXQ);}

S(2)

{TS.Chain=Merge(T,S(2).Chain);}

意义及功能：读到then时,应以当前NXQ值回填RE的T链（真出口）; 在扫视到else时则对F链（假出口）的回填。最后将S(1),S(2)产生的后继语句合成一个链表，放到TS链（指示TS程序结构全部出口构成的链，有待于在适当的时机回填）。

（3）循环语句：WS->WHILE RE DO S

动作解释：

WS->WHILE

{LoopS=NewTemp();LoopS=NXQ;}

RE DO

{BackPatch($2.TC,NXQ);

WChain=NewTemp();WChain=$2.FC;

WLoopS=NewTemp();WLoopS=LoopS;}

S

{BackPatch($2.Chain,WLoopS);

GEN(j,0,0,WLoopS);

$$.Chain=WChain;}

意义及功能： 读到while时，将NXQ（第一个四元式序号）赋给临时变量loopS，以便翻译完S(1)后产生转向此处的四元式。读到do时，当前NXQ值回填RE的T链（真出口）,将RE的F链（假出口，整个循环语句的出口）存在临时变量WChain中，并将上一个loops存在临时变量WLoopS里，跳到S时，即将本结构的第一个四元式WLoopS属性回填到RE链中。接着产生一个以此序号为转向目标的无条件四元式，作为整个WS的属性保留下来，待适当机会回填。

（4）AE->AE+TM (中间符号可以是任意四则运算符)

{$$.PLACE=NewTemp();

GEN(+,$1.PALCE,$3.PALCE,$$.PALCE);}

（5）F->(AE)

{$$.PLACE=$2.PALCE;}

（6）VI->ID

{$$.PALCE=Entry($1)}

（7）(5) RE->AE RS AE

{$$.TC=NXQ;$$.FC=NXQ+1;

GEN(jrop,Entry($1),Entry($3),0);

GEN(j,0,0,0);}

（8）S->Assignment

{$$.Chain=0;}

（9）ST->S;ST

{BackPatch($1.Chain,NXQ);}

（11）CS->BEGIN ST END

{$$.Chain=$2.Chain;}

## 5.2 语义分析程序设计

5.2.1设计结构体和主要变量（如表5-1所示）

表5-1 结构体和主要变量表

|  |  |
| --- | --- |
| **结构体/变量名称** | **作用** |
| struct formula gen[MAXSIZE] | 四元式结构体，通过对该栈的操作，实现操作码、操作数等信息的更新。 |
| char \*tempvar[MAXSIZE] | 存放临时变量表，辅助产生四元式。 |
| int TS | 存放栈顶状态号。 |
| int IS | 存放当前输入符号。 |
| int NS | 存放规约后GOTO产生的新状态。 |
| int PL | 存放第i个产生式的左部(编号)。 |

5.2.2 主要方法及作用（如表5-2所示）

表5-2 主要方法和作用表

|  |  |
| --- | --- |
| **方法名称** | **作用** |
| int Entry（char \*buf） | buf为名查符号表，若buf存在符号表内则返回其所在位置的下标，找不到则填入符号表。 |
| int NewTemp() | 申请临时工作单元，一般用来存放运算结果。 |
| int Merge(int p1,int p2) | 将链首“指针”分别为p1和p2的两条链合并为一条，并返回新链的链首”指针。 |
| void BackPatch(int p,int t) | 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。 |
| void act(stack \*st,int num) | 执行归约产生式对应的语义动作。 |

## 5.3 主要模块的算法

int NewTemp() //申请临时工作单元 （一般存放运算结果）

{

int num;

string[string\_max]=tempvar[tempvar\_cur];

num=string\_max;

string\_max++;

tempvar\_cur++;

return num;

/\*产生临时变量的函数，每次调用都定义一个新的临时变量。返回值为该变量的编号\*/

}

/\*将链首“指针”分别为p1和p2的两条链合并为一条，并返回新链的链首”指针“\*/

int Merge(int p1,int p2)

{

int p;

if(!p2)

return p1;

else

{

p=p2;

while(gen[p].result)

p=gen[p].result;

gen[p].result=p1;

return p2;

}

}

//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

void BackPatch(int p,int t)

{

int q1;

int q=p;

while(q)

{

q1=gen[q].result;

gen[q].result=t;

q=q1;

}

}

void act(stack \*st,int num) //执行归约产生式对应的语义动作

{

if(num==6||num==7||num==28) //THEN ELSE ：

{

production[ACTION[TS][IS].s].Pro\_place=Entry(st->ch\_word[st->top]);

}

else if(num==5||num==8||num==9||num==23||num==24||num==25)

{

production[ACTION[TS][IS].s].Pro\_place=st->st\_place[st->top];

}

else if(num==10) //int

{

tempstr=(char \*)malloc(3\*sizeof(char));

tempstr=":=";

gen[NXQ].opr=tempstr;

gen[NXQ].op1=st->ch\_word[st->top-2];

gen[NXQ].op2=string[st->st\_place[st->top]];

gen[NXQ].result=0;

NXQ++;

}

else if(num==11) //var

{

production[ACTION[TS][IS].s].Pro\_place=NewTemp();

tempstr=(char \*)malloc(2\*sizeof(char));

tempstr="+";

gen[NXQ].opr=tempstr;

gen[NXQ].op1=string[st->st\_place[st->top-2]];

gen[NXQ].op2=string[st->st\_place[st->top]];

gen[NXQ].results=string[production[ACTION[TS][IS].s].Pro\_place];

NXQ++;

}

else if(num==14) //\*

{

production[ACTION[TS][IS].s].Pro\_place=st->st\_place[st->top-1];

}

else if(num==15) // /

{

BackPatch(st->st\_TC[st->top],st->st\_FC[st->top-1]);

gen[NXQ].opr="j";

gen[NXQ].op1="0";

gen[NXQ].op2="0";

gen[NXQ].result=st->st\_FC[st->top-1];

NXQ++;

production[ACTION[TS][IS].s].Pro\_TC=st->st\_TC[st->top-1];

}

else if(num==17) //<=

{

production[ACTION[TS][IS].s].Pro\_TC=NXQ;

production[ACTION[TS][IS].s].Pro\_FC=NXQ+1;

gen[NXQ].opr=string[st->st\_place[st->top-1]];

gen[NXQ].op1=string[st->st\_place[st->top-2]];

gen[NXQ].op2=string[st->st\_place[st->top]];

gen[NXQ].result=0;

NXQ++;

gen[NXQ].opr="j";

gen[NXQ].op1="0";

gen[NXQ].op2="0";

gen[NXQ].result=0;

NXQ++;

}

else if(num==18) //>

{

production[ACTION[TS][IS].s].Pro\_TC=Merge(T,st->st\_FC[st->top]);

}

else if(num==20) //<>

{

BackPatch(st->st\_TC[st->top-2],NXQ);

gen[NXQ].opr="return";

gen[NXQ].op1="0";

gen[NXQ].op2="0";

gen[NXQ].result=0;

NXQ++;

gen[NXQ].result=99;

printf("success");

}

else if(num==22) //;

{ production[ACTION[TS][IS].s].Pro\_TC=0; }

}

# 6 系统结果及分析

## 6.1 测试案例

**程序1（符合Pascal语法规则）：**

VAR

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

BEGIN//这是注释

WHILE (A+1)>B DO

IF C>D THEN

X:=Y+Z

ELSE

Y:=X+Z

END.

**程序2（不符合Pascal词法规则）：**

VAR @

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

BEGIN

WHILE (A+ 65342171)>B DO

IF C>D THEN

X:=Y+Z

ELSE

Y:=X+Z

END.

**程序3（不符合Pascal语法规则）：**

VAR

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

BEGIN

WHILE A+1 DO //该行不符合语法规则

X:=X+1

END.

## 6.2 词法分析结果

6.2.1正确用例：

结果显示，没有错误，且注释的内容已忽略，产生二元式如图6-1所示：

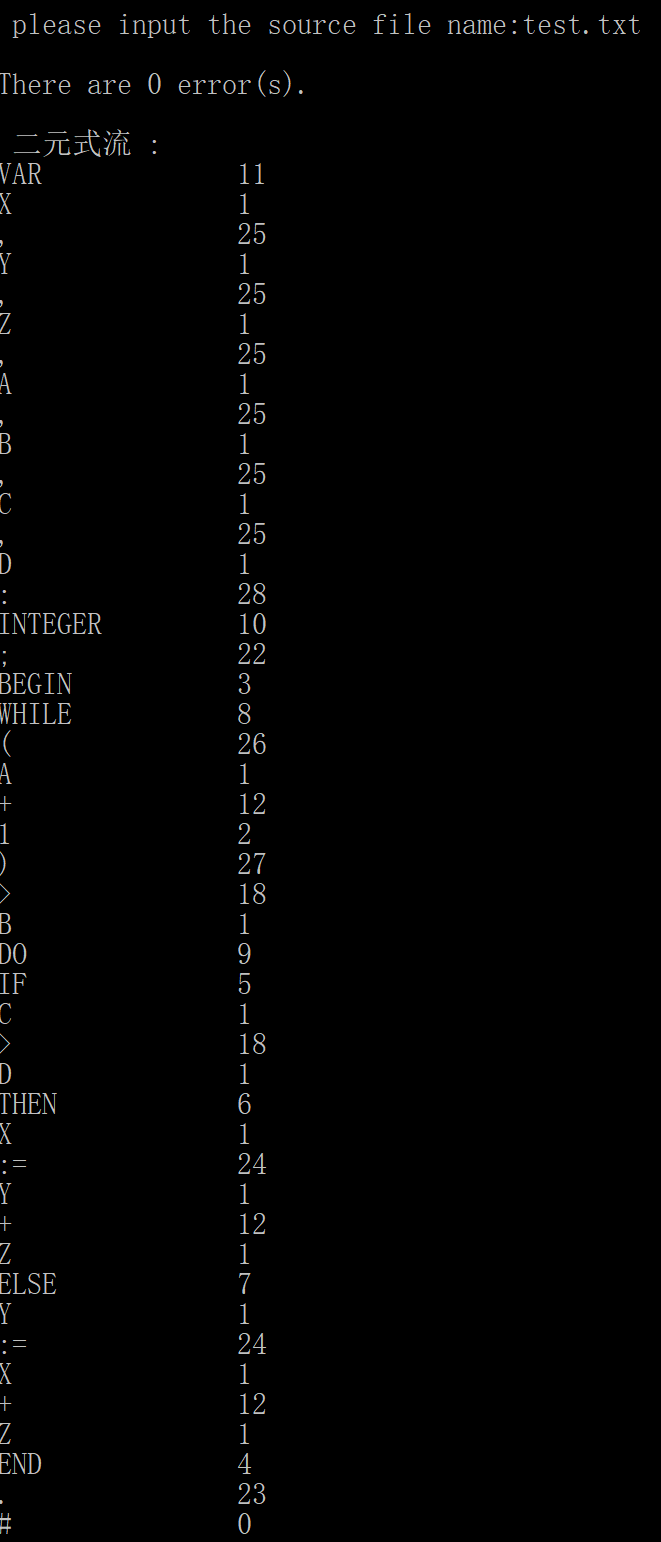


图6-1 实验结果1

6.2.2 错误用例：

如图6-2显示，第一行错误原因为非法字符‘@’，第二行为标识符错误‘fdy’，第四行为字符串长度溢出。

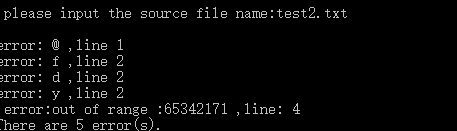


图6-2 实验结果2

## 6.3 语法分析结果

运行结果如图6-3、6-4、6-5所示。

### 

图 6-3 语法分析运行结果（1）



图 6-4 语法分析运行结果（2）

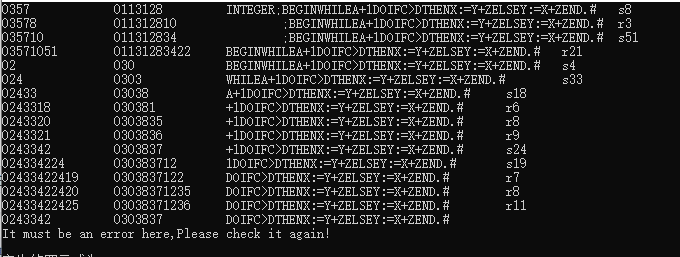


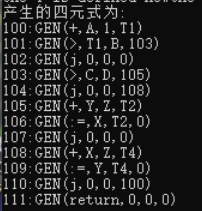
图 6-5 语法分析运行结果（3）

语法分析部分运行结果如上图，其中图6-3、6-4 是运行程序一（正确程序）的截图；图6-5是运行程序二（错误程序）的截图。

由上面语法分析的最一行的Accepted!可以知道程序语法分析结束，该程序的语法符合要求。而由图6-5也给出了错误提示。并结束后面的语法分析。

## 6.4 语义分析结果

运行结果如6-6所示。

 图 6-6 语义分析翻译结果

生成的四元式如上图所示，进入程序后，A和1做加法运算赋值给T1，若T1>B，则跳转到103，执行该序号对应的四元式；否则往下走到102并结束该程序。执行103时，进入if-then语句的判断，若C>D则跳转到105执行Y和Z做加法运算赋值给T2，并往下走到106将T2的值赋给X，接着走到107结束该程序；否则跳转到108执行X和Z做加法运算赋值给T4。往下走到109将T4的值赋给Y，接着重新跳回开头的While语句判断，重复以上动作，直到不符合条件跳出该程序。

# 7 总结

在进行词法分析时，我处理双目运算符的时候遇到了困难，因为采用C语言，采用优先判断双目运算符，多取出一个符号，构造2个符号的字符串变量查编号表，如果查不到，则不是双目运算符，应该文件指针回退，判断是否为单目运算符。

在实验中，一开始在查表的时候，不能判别当前程序是第几行，后来在初始化init()函数中把各个字符分类写入文件保存时，每写入一个字符串，就在后面多输出一个换行符“\n”，这样在查表的时候，通过判断“\n”的个数来判断程序当前所在的行。这样就解决了标记行数的问题。

在进行语法分析的实现时，的确遇到了不少问题，在解决这些问题的同时，也提高了我的相关知识面。例如在最初时因为考虑到程序单词数未知，所以计划采用链表来模拟栈，但是在实际使用时，发现链表出栈相对麻烦，所以改用结构体来模拟栈，里面的状态栈、符号栈采用数组来模拟，用一个整型变量来记录栈顶。还有一个最基础的就是由于语法分析的过程输出时没有考虑元素长度问题，直接输出，导致界面十分不美观，后面测试时发现了这一问题，对输出内容格式修整后再输出就解决了这一问题。

在进行语义分析时，由于对这块知识点掌握的不透彻，经过网上查找资料，还有阅读图书馆借阅到的关于语义分析的书籍，终于对语义分析的流程和概念有了一定的了解和掌握。但是由于个人编写代码能力的薄弱，在实现过程中经过多次调试才能解决大部分问题，仍存在小部分问题无法解决。在条件语句和循环语句的翻译遇到了困难，因为语法分析时没有进行拆分，而是到整个句子结结尾才进行归约，所以只有进行分步执行语义动作，这样的后果是不仅仅是归约的时候产生语义动作，而且移进的时候也要执行相应的语义动作使得程序结构比较乱，易读性差。并且语义分析到while语句里的if-then语句时，真出口没有回填刚开始的假链，个人能力有限，没能调试出来。

以前只会用SLR分析法进行做题，现在明白了SLR分析法的原理，明白了它的高明之处。在进行语法分析时，难点是构造SLR分析表，一旦分析表构造成功，语法分析则一蹴而就。接下来困难之处是调试阶段。刚开始只能单步调试，随后经过改进，加入输出函数，输出分析的过称，这样能很清楚的看到那里出现错误，很方便调试。

由于语法分析与词法分析相对独立，只需要基于词法分析得到的结果(token.txt）来进行语法分析即可。在队友进行词法分析时，我便开始着手实现语法分析，提高了我们小组完成该课程设计的效率。

通过这次课程设计，我对词法分析、语法分析和语义分析都有了更深的理解。此外，对团队分工合作以提高效率有了更直观的感受，在具体实现功能的过程中，我的动手能力得到了提高的同时，也进一步了解了编译原理的相关模块的内容的实际，对其实际意义和作用有了更深刻的理解，可以说收获颇多。

参考文献

[1] 陈意云.编译原理和技术.中国科技大学出版社，1997

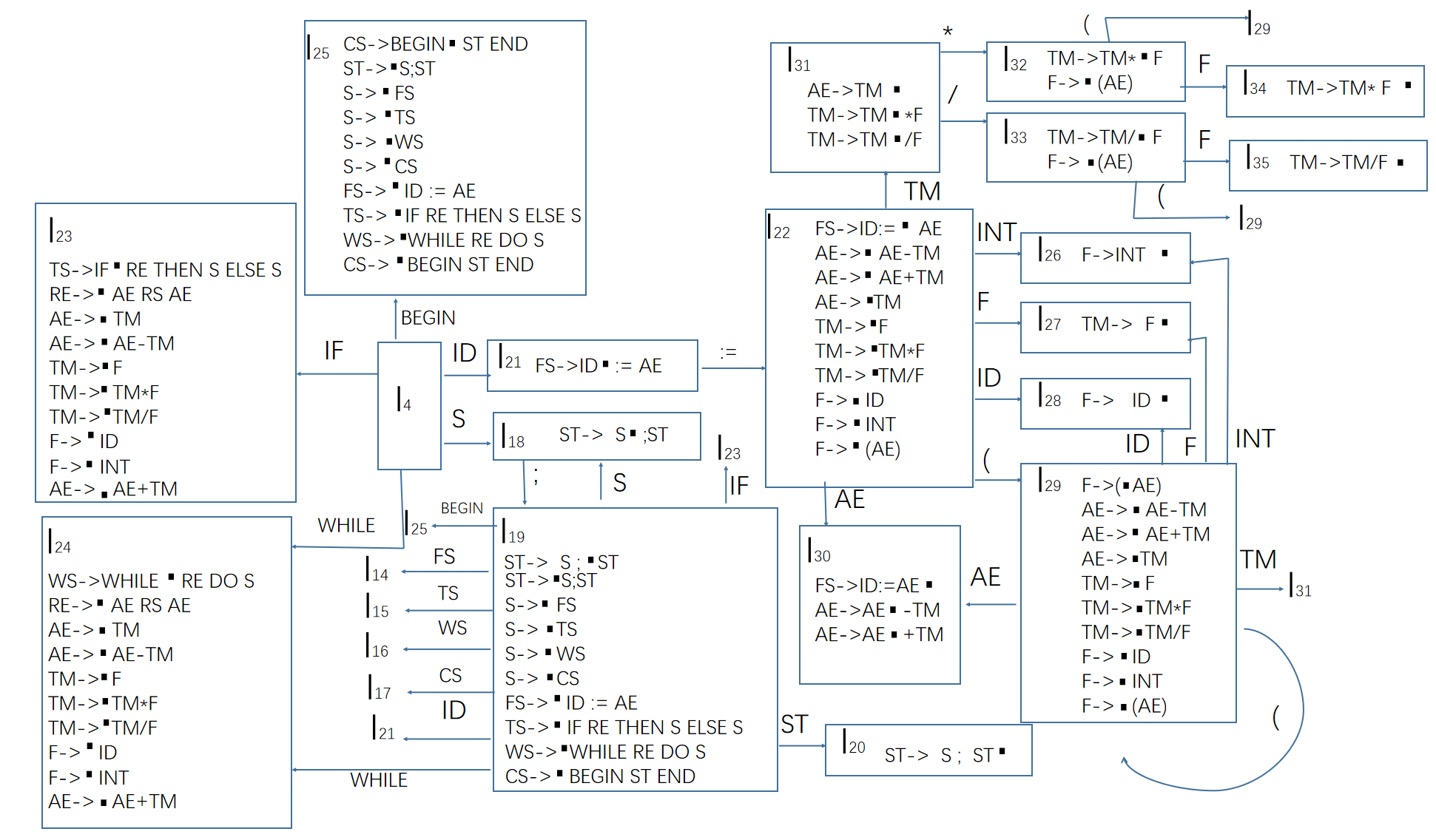
[2] 杜淑敏，王永宁.编译程序设计原理.北京大学出版社，1986.

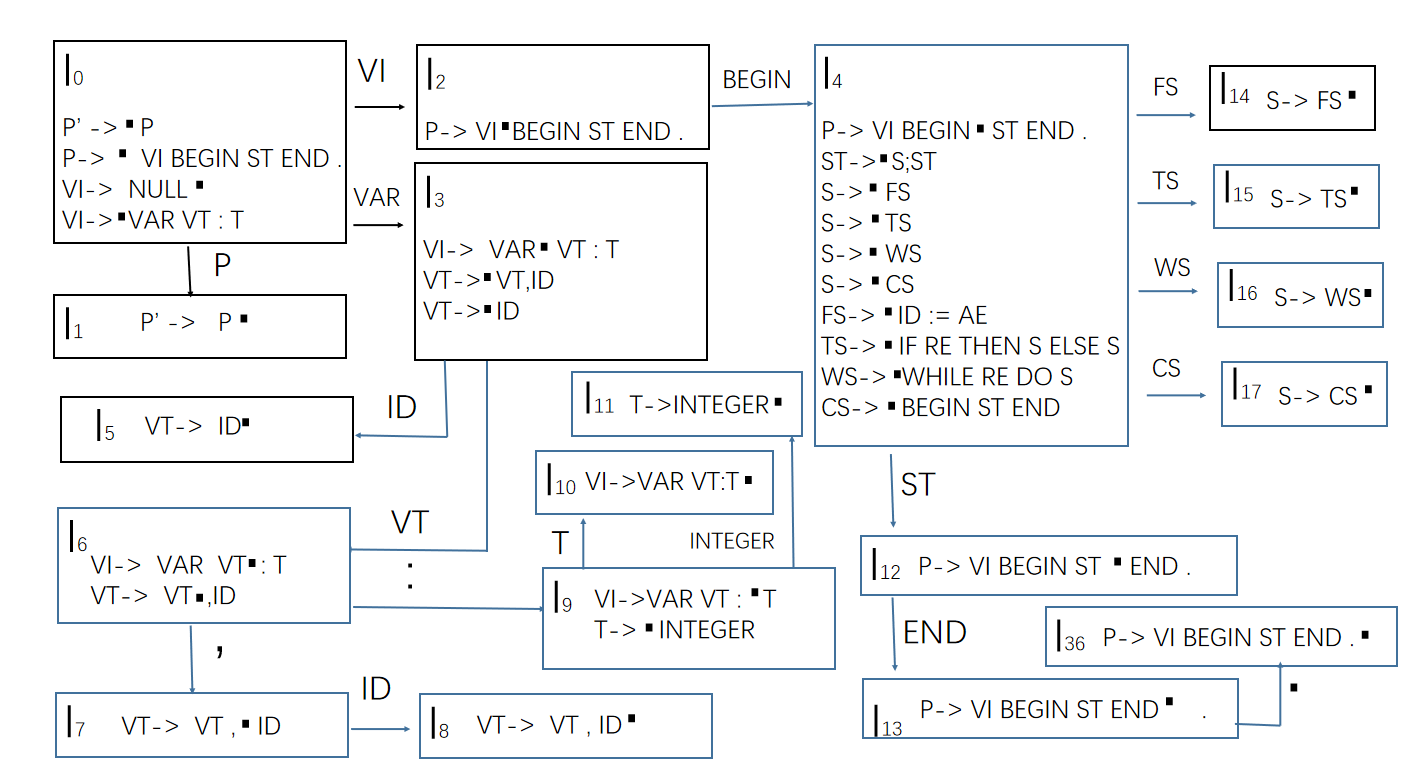
[3] 金成植.编译程序构造原理和实现技术.高等教育出版社，2000.

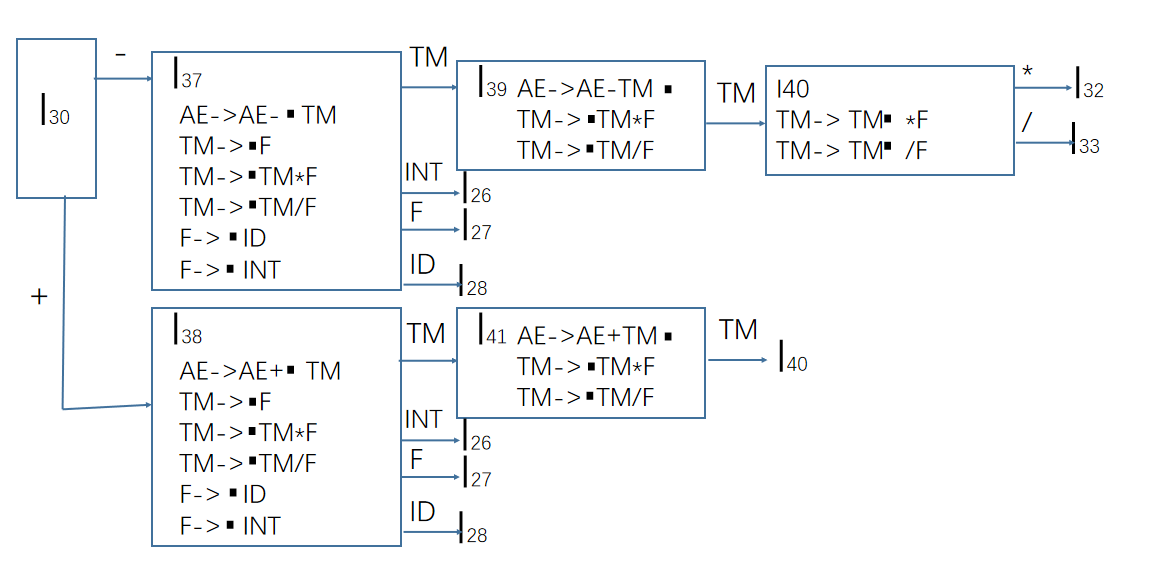
[4] 徐宝文，李志译，尤晋元. C 程序设计语言. 机械工业出版社，2004.

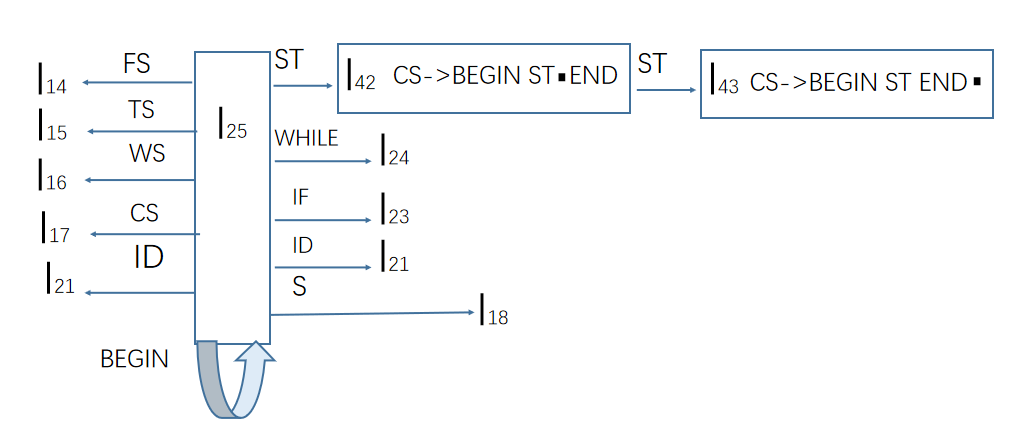
附录一

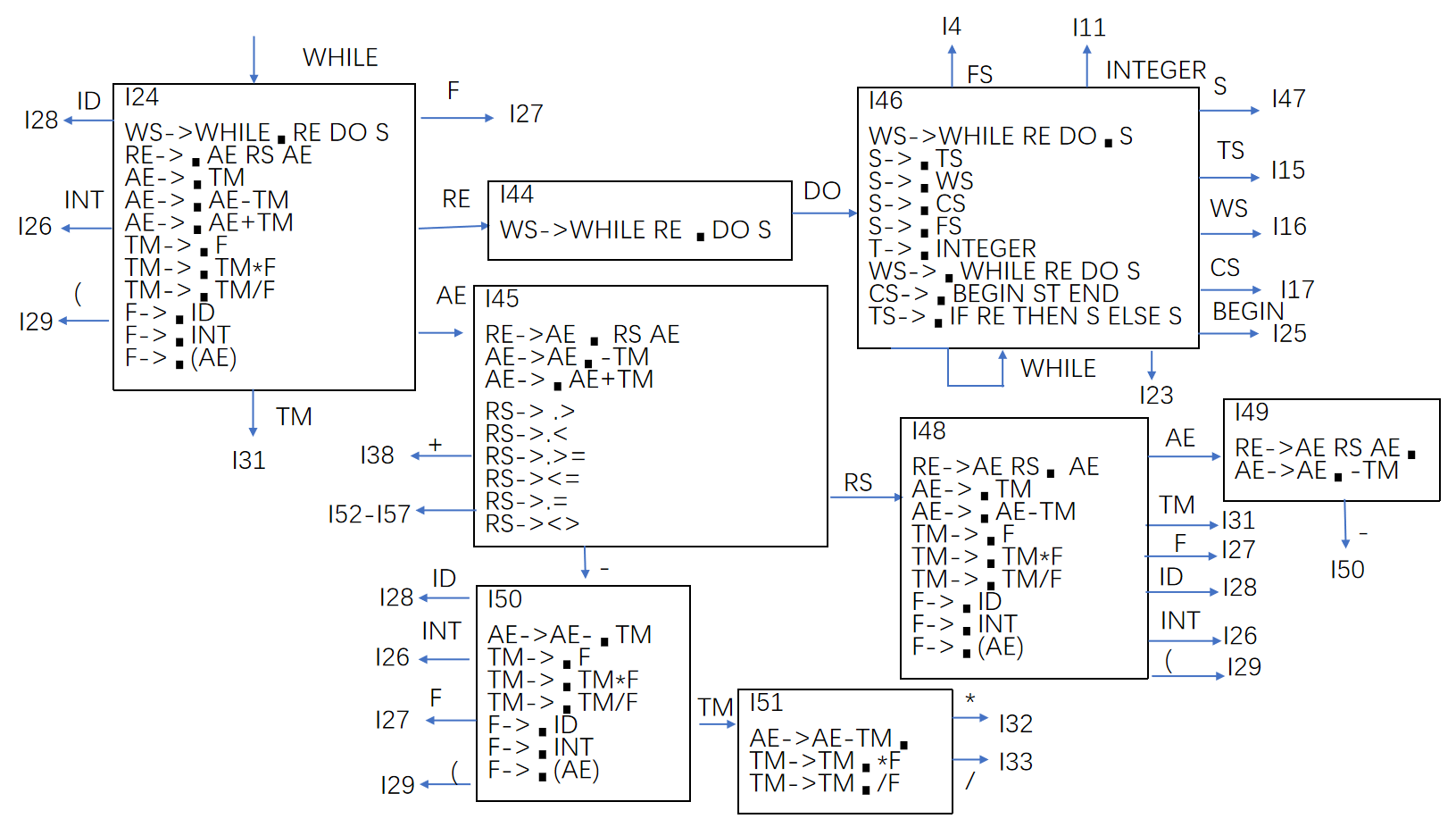
DFA构造表

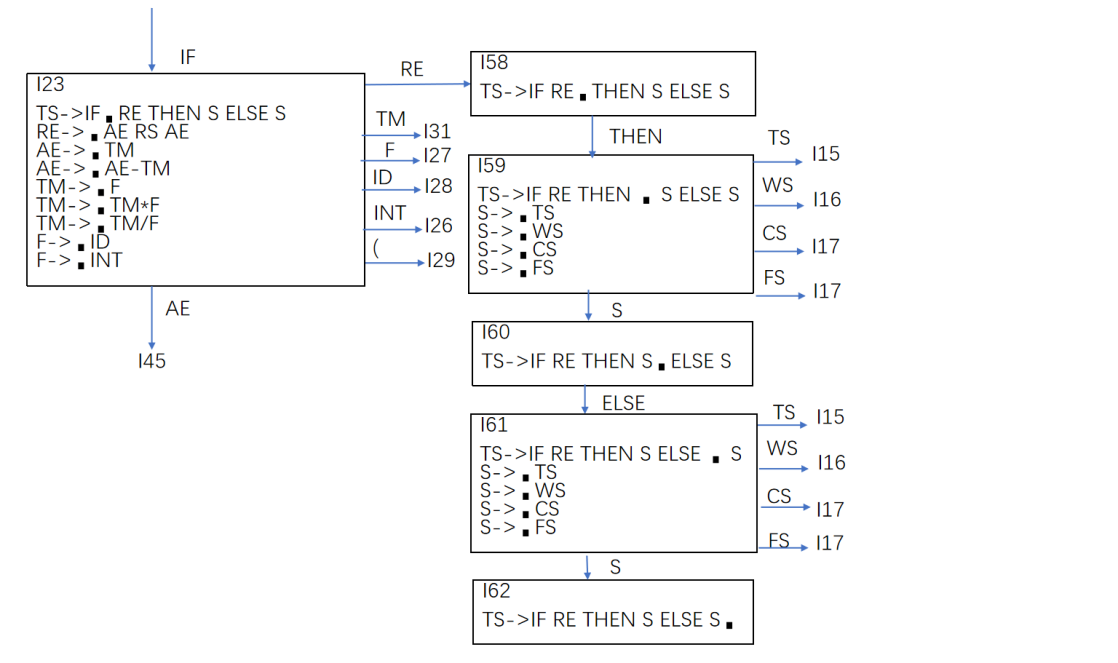












附录二

SLR文法分析表

