廣東工業大學

编译原理实验报告

课程名称_____编译原理实验_____

学生学院_____计算机学院_____

专业班级_____网络工程 1 班_____

学　　号_____3116004932_____

学生姓名_____郑臣河_____

指导教师_____蒋艳荣_____

程序功能完成情况	
测试用例全面程度	
代码开发风格、用户界面	
报告格式是否与要求相符	
报告内容是否准确、详实	

2019 年　7　月　 3　日

目 录

1 概述

1.1 实验内容

1)必做内容：

对 PL/0 作以下修改和扩充，并使用测试用例验证：

（1）修改单词：不等号# 改为 != ，只有！符号为非法单词，同时#成为非法符号。

（2）增加单词(只实现词法分析部分)：

保留字 ELSE，FOR，STEP，UNTIL，DO,RETURN

运算符 *=，/=，&，||

注释符 //

（3）增加条件语句的 ELSE 子句（实现语法语义目标代码），

要求：写出相关文法和语法图，分析语义规则的实现。

2）选择内容 1：

（1）扩充赋值运算：*= 和 /=

(2)扩充语句（Pascal 的 FOR 语句）：

FOR <变量>:=<表达式>STEP<表达式> UNTIL<表达式>Do<语句>

此次实验，完成了必做的三个实验，以及两个选择实验

1.2 实验环境

源语言：PL/0 语言

目标语言：类 P-Code

实现工具：C++ builder

运行平台：win10

2 结构设计说明

2.1 各功能模块描述

pl0	主程序
error	出错处理，打印出错位置和错误编码
getsym	词法分析，读取一个单词
getch	漏掉空格，读取一个字符
gen	生成目标代码，并送入目标程序
test	测试当前单词符号是否合法
block	分析程序分析处理过程
enter	登陆名字表
position	查找标识符在名字表中的位置
constdeclaration	常量定义处理
vardeclaration	变量说明处理
listcode	列出目标代码清单
statement	语句处理
expression	表达式处理
term	项处理
factor	因子处理
condition	条件处理
interpret	对目标代码的解释执行程序
base(函数)	通过静态链求出数据区的基地址

3 主要成分描述

3.1 1.符号表

为了组成一条指令，编译程序必须知道其操作码及其参数(数或地址)。这些 值是由编译程序本身联系到相应标识符上去的。这种联系是在处理常数、变量和 过程说明完成的。为此，标识符表应包含每一标识符所联系的属性；如果标识符 被说明为常数，其属性值为常数值；如果标识符被说明为变量，其属性就是由层 次和修正量(偏移量)组成的地址；如果标识符被说明为过程，其属性就是过程的 入口地址及层次。常数的值由程序正文提供，编译的任务就是确定存放该值的地 址。我们选择顺序分配变量和代码的方法；每遇到一个变量说明，就将数据单元 的下标加一(PL/0 机中，每个变量占一个存储单元)。开始编译一个过程时，要对 数据单元的下标 dx 赋初值，表示新开辟一个数据区。dx 的初值为 3，因为每个 数据区包含三个内部变量 RA,DL 和 SL。

3.2 2.运行时存储组织和管理

对于源程序的每一个过程(包括主程序)，在被调用时，首先在数据段中开辟 三个空间，存放静态链 SL、动态链 DL 和返回地址 RA。静态链记录了定义该过 程的直接外过程(或主程序)运行时最新数据段的基地址。动态链记录调用该过程 前正在运行的过程的数据段基址。返回地址记录了调用该过程时程序运行的断点 位置。对于主程序来说，SL、DL 和 RA 的值均置为 0。静态链的功能是在一个 子过程要引用它的直接或间接父过程(这里的父过程是按定义过程时的嵌套情况 来定的，而不是按执行时的调用顺序定的)的变量时，可以通过静态链，跳过个数 为层差的数据段，找到包含要引用的变量所在的数据段基址，然后通过偏移地址 访问它。 在过程返回时，解释程序通过返回地址恢复指令指针的值到调用前的地址， 通过当前段基址恢复数据段分配指针，通过动态链恢复局部段基址指针。实现子 过程的返回。对于主程序来说，解释程序会遇到返回地址为 0 的情况，这时就认 为程序运行结束。 解释程序过程中的 base 函数的功能，就是用于沿着静态链，向前查找相差 指定层数的局部数据段基址。这在使用 sto、lod、stoArr、lodArr 等访问局部变量 的指令中会经常用到。 类 PCODE 代码解释执行的部分通过循环和简单的 case 判断不同的指令，做 出相应的动作。当遇到主

程序中的返回指令时，指令指针会知道 0 位置，把这样 一个条件作为终止循环的条件，保证程序运行可以正常地结束。

3.3 3.语法分析方法

语法分析子程序采用了自顶向下的递归子程序法，语法分析同时也根据程序 的语意生成相应三元代码，并提供了出错处理的机制。语法分析主要由分程序分 析过程(BLOCK)、参数变量分析过程(ParaDeclaration)、参数变量处理过程 (ParaGetSub)、数组处理过程(ParaGetSub)、常量定义分析过程(ConstDeclaration)、 变量定义分析过程(Vardeclaration)、语句分析过程(Statement)、表达式处理过程 (Expression)、项处理过程(Term)、因子处理过程(Factor)和条件处理过程(Condition) 构成。这些过程在结构上构成一个嵌套的层次结构。除此之外，还有出错报告过 程(Error)、代码生成过程(Gen)、测试单词合法性及出错恢复过程(Test)、登陆名字表过程(Enter)、查询名字表函数(Position)以及列出类PCODE代码过程(Listcode) 作语法分析的辅助过程。

3.4 4.中间代码表示

中间代码是源程序的一种内部表示，复杂性介于源语言和目标机语言之间。

中间代码的表示方法有逆波兰式、三元式、树形、四元式等。(1) 逆波兰记号是最简单的一种中间代码表示形式，早在编译程序出现之前，它 就用于表示算数表达式。后缀表示法表示表达式，其最大的有点是易于栈式 计算机处理表达式。(2) 每个三元式由三个部分组成：A.算符 op；B.第一运算对象 ARG1；C.第二运 算对象 ARG2；运算对象可能是源程序中的额变量，也可能是某个三元式的 结果，用三元式的编号表示。(3) 树形表示是三元式表示的翻版 (4) 四元式是一种比较普遍采用的中间代码形式：算符 op，运算对象 ARG1，运 算对象 ARG2，运算结果 RESULT

4 测试用例

4.1 必做

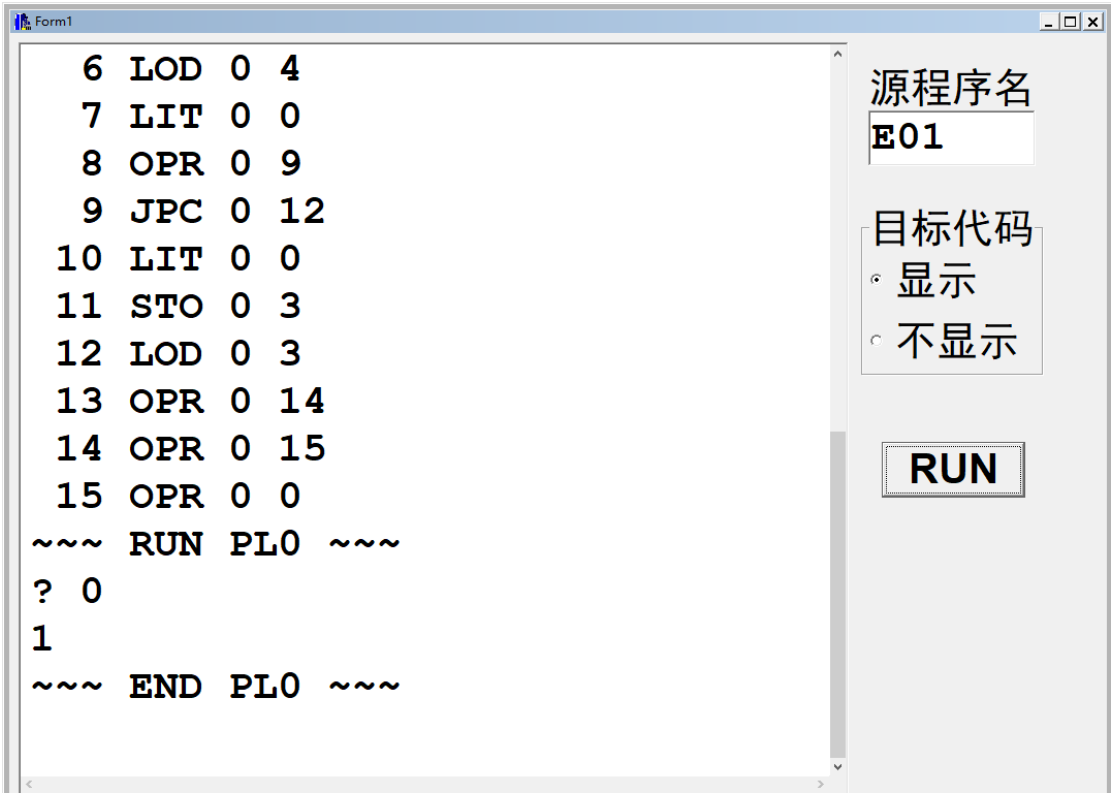
4.1.1 实验 1

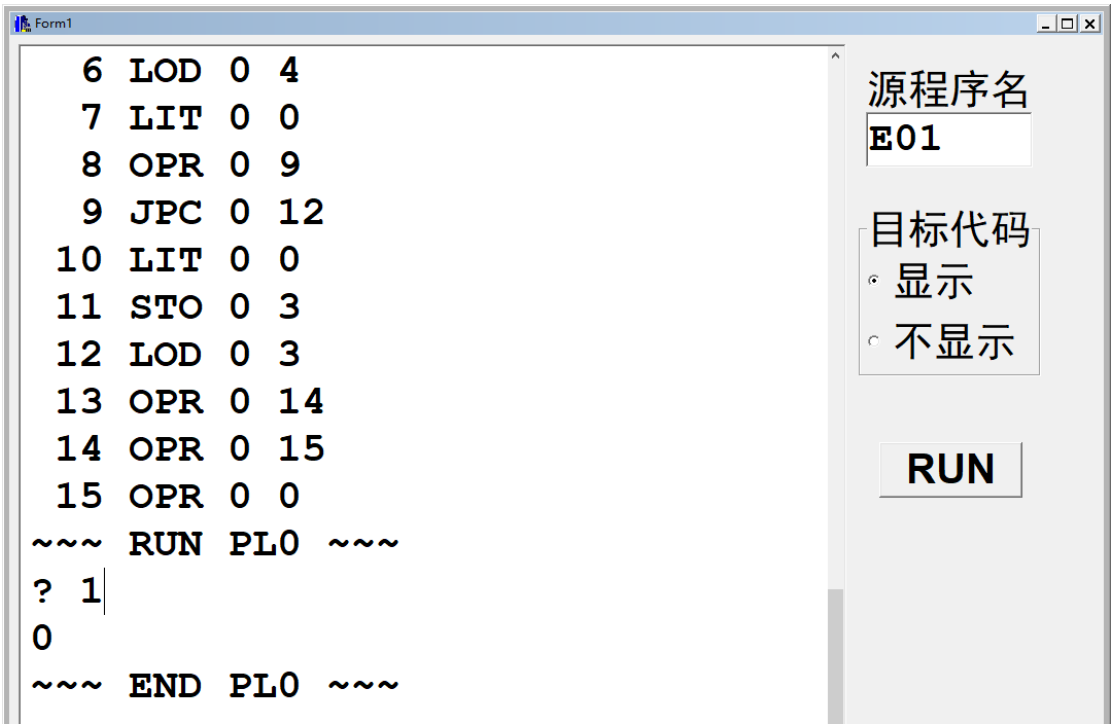
测试代码:

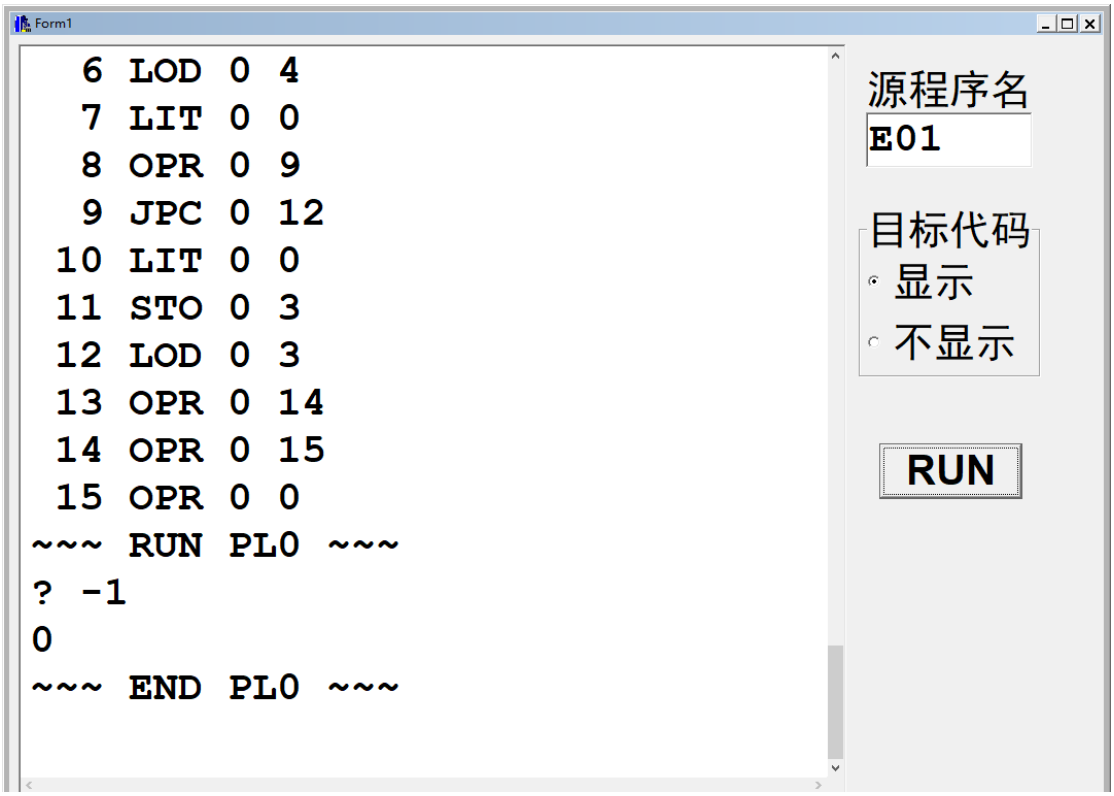
```
PROGRAM EX01;
VAR A,B;
BEGIN
   A:=1;
   READ(B);
   IF B != 0 THEN A:=0;
   WRITE(A);
END.
```

功能描述	识别!=号	
用例目的	将’!=’识别为不等于并将只有’！’、’#’为非法单词	
前提条件	无	
输入/动作	期望输出	实际情况
B = 0	窗口输出 1	窗口输出 1
B = 1	窗口输出 0	窗口输出 0
B = -1	窗口输出 0	窗口输出 0

测试截图如下图所示


Form1
6 LOD 0 4
7 LIT 0 0
8 OPR 0 9
9 JPC 0 12
10 LIT 0 0
11 STO 0 3
12 LOD 0 3
13 OPR 0 14
14 OPR 0 15
15 OPR 0 0
~~~ RUN PL0 ~~~
? 0
1
~~~ END PL0 ~~~
源程序名
E01
目标代码
显示
不显示
RUN


Form1
6 LOD 0 4
7 LIT 0 0
8 OPR 0 9
9 JPC 0 12
10 LIT 0 0
11 STO 0 3
12 LOD 0 3
13 OPR 0 14
14 OPR 0 15
15 OPR 0 0
~~~ RUN PL0 ~~~
? 1
0
~~~ END PL0 ~~~
源程序名
E01
目标代码
显示
不显示
RUN



PROGRAM EX01;

VAR A;

BEGIN

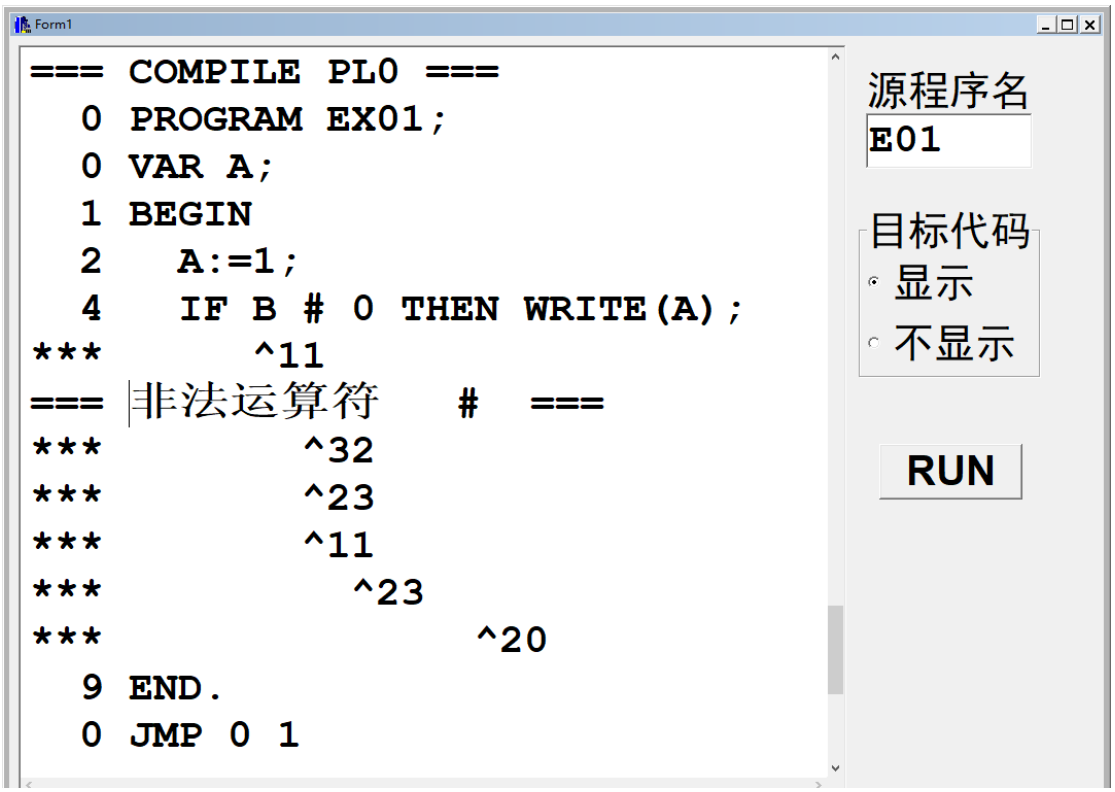
A:=1;

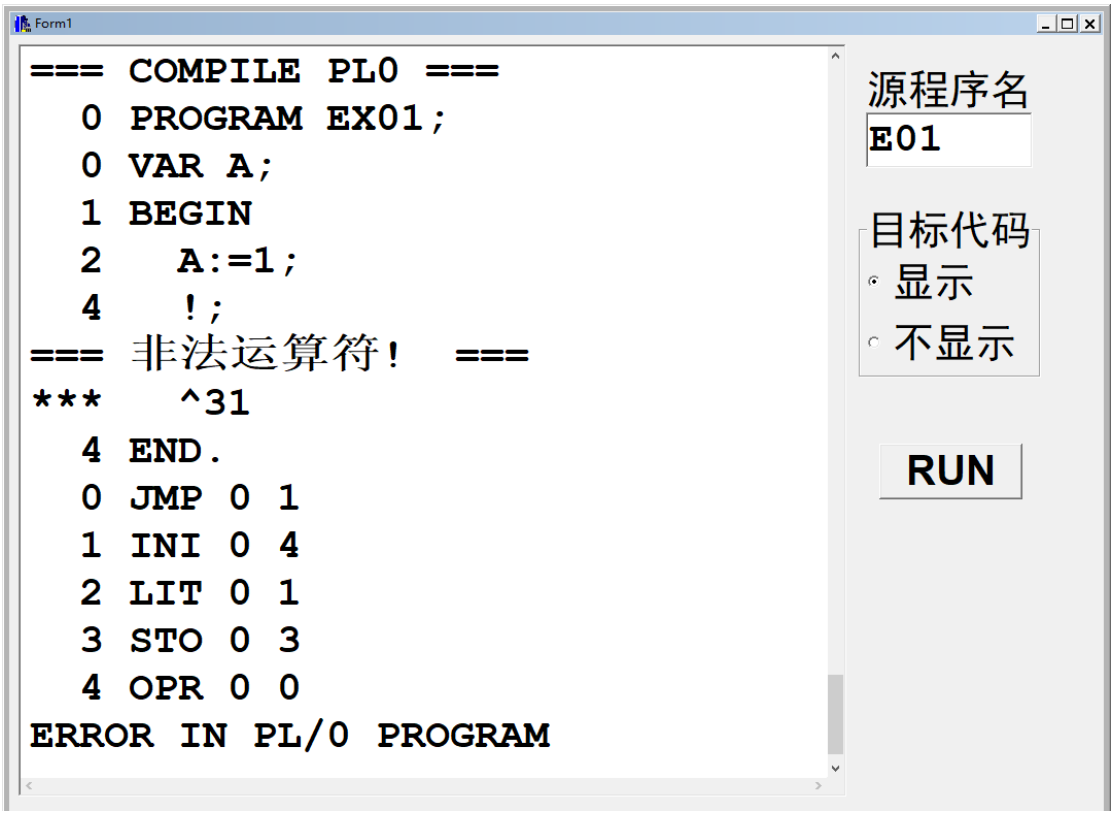
(4)

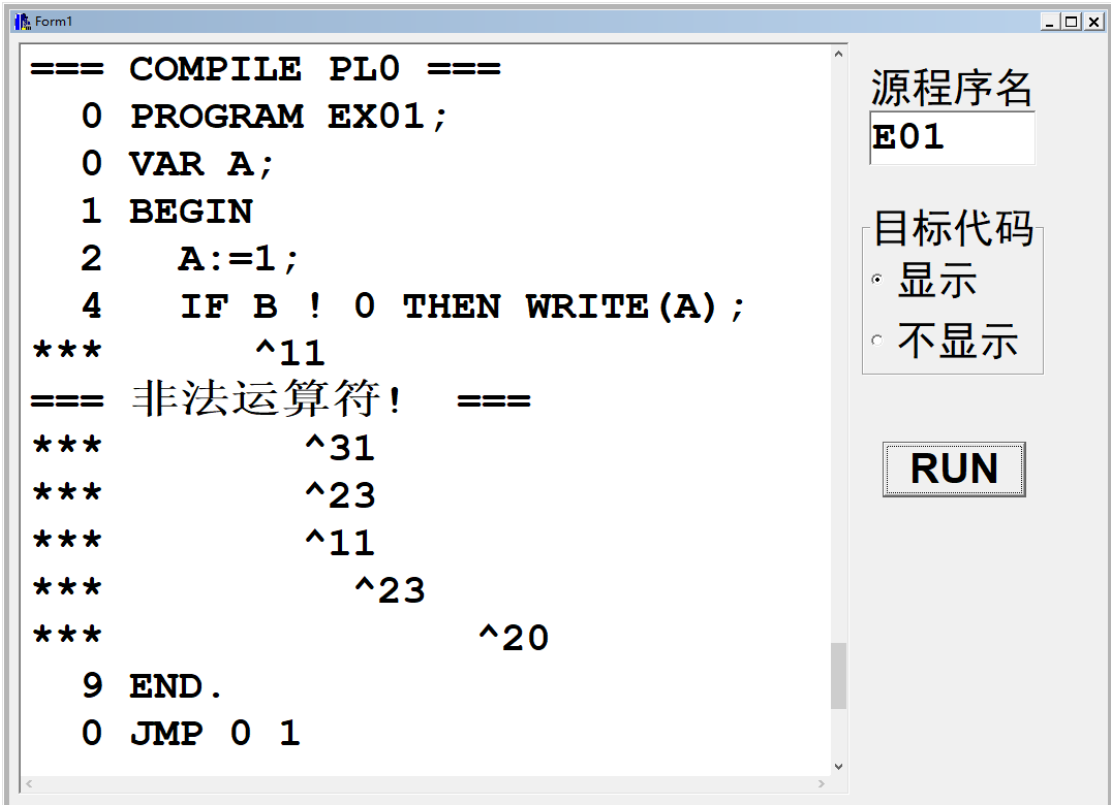
END.

功能描述	识别!=号	
用例目的	将只有’！’、’#’为非法单词	
前提条件	无	
输入/动作	期望输出	实际情况
(4) = IF B # 0 THEN WRITE(A);	报错 非法运算符#	非法运算符 #
(4) = !;	报错 非法运算符!	非法运算符 ！
(4) = IF B ! 0 THEN WRITE(A);	报错 非法运算符!	非法运算符 ！

测试截图如下：


Form1
=== COMPILE PL0 ===
0 PROGRAM EX01;
0 VAR A;
1 BEGIN
2 A:=1;
4 IF B # 0 THEN WRITE(A);
*** ^11
=== 非法运算符 # ===
*** ^32
*** ^23
*** ^11
*** ^23
*** ^20
9 END.
0 JMP 0 1
源程序名
E01
目标代码
显示
不显示
RUN


Form1
=== COMPILE PL0 ===
0 PROGRAM EX01;
0 VAR A;
1 BEGIN
2 A:=1;
4 !;
=== 非法运算符! ===
*** ^31
4 END.
0 JMP 0 1
1 INI 0 4
2 LIT 0 1
3 STO 0 3
4 OPR 0 0
ERROR IN PL/0 PROGRAM
源程序名
E01
目标代码
显示
不显示
RUN



实现该部分实验的功能的核心是词法分析，即修改 GetSym 函数。GetSym 函数会忽略空格每次读取一个词，再对词进行识别（保留字、分隔符、变量、操作符）。识别 != 得代码如下：

```
else if(CH=='!'){
        GetCh();
        if(CH=='='){
            SYM=NEQ; // 该语句表示 识别 != 识别为保留的操作符
            Form1->printfs("=== 识别!= ===");
            GetCh();
        }else{
            Form1->printfs("=== 非法运算符! ===");
            Error(31);
        }
    }
```

并且我们需要把单独得 # 识别为非法,代码如下：

```
    }else if(CH=='#'){
```

```
        Form1->printfs("=== ·Ç·¨ÔËËã·û¬¦ #    ===");
                GetCh();
        Error(32);
    }
```

4.1.2 实验 2

增加单词(只实现词法分析部分)：

保留字 ELSE，FOR，STEP，UNTIL，TYPEDEF

运算符 *=，/=

注释符 //

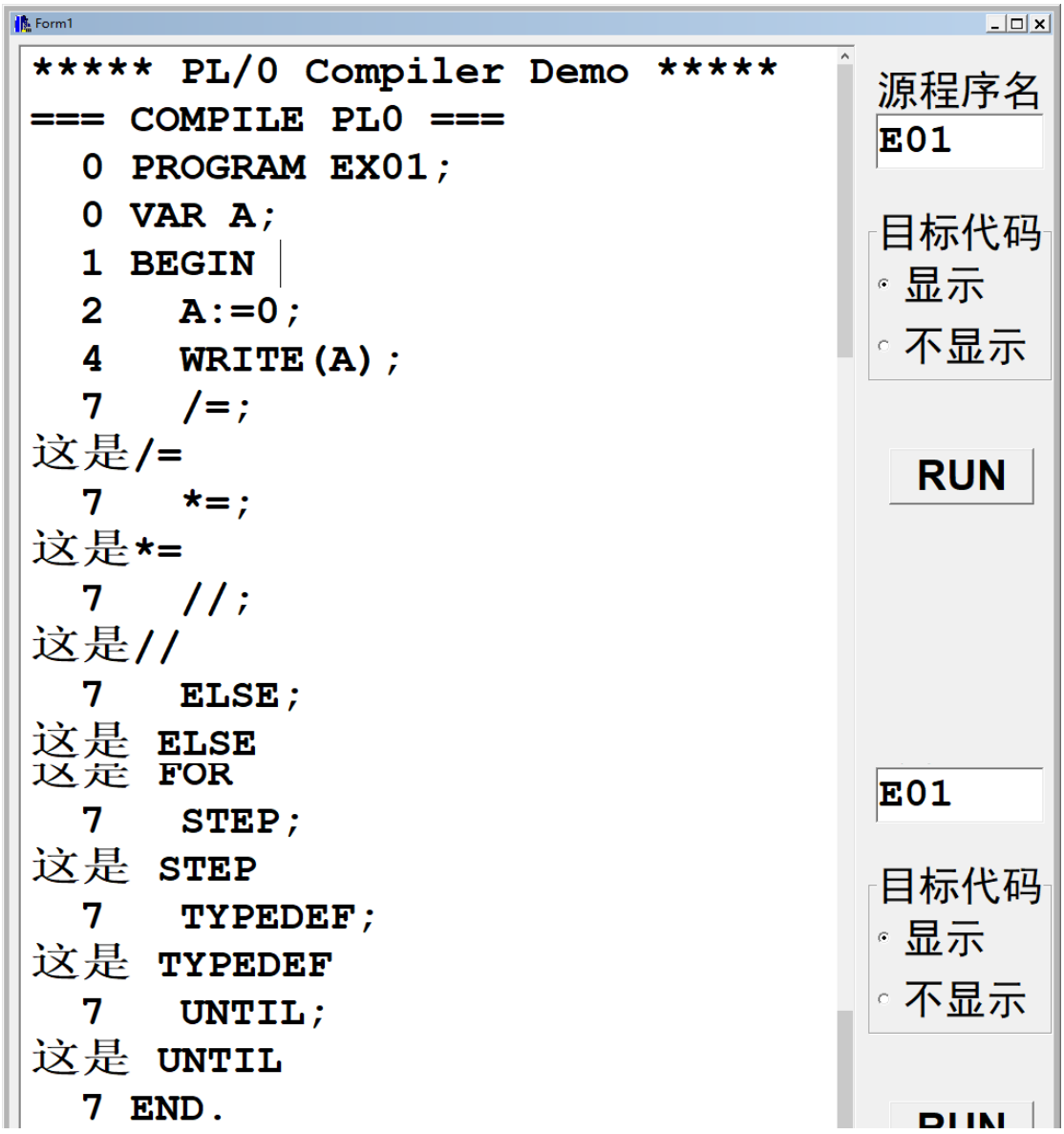
测试代码：

```
PROGRAM EX01;
VAR A;
BEGIN
  A:=0;
  WRITE(A);
  /=;
  *=;
  //;
  ELSE;
  FOR;
  STEP;
  TYPEDEF;
  UNTIL;
END.
```

功能描述	识别单词(只实现词法分析部分)

用例目的	识别 ELSE，FOR，STEP，UNTIL，TYPEDEF *=，/= //		
前提条件	无		
输入/动作		期望输出	实际情况
E1SE		这是 E1SE	这是 E1SE
FOR		这是 FOR	这是 FOR
STEP		这是 STEP	这是 STEP
UNTIL		这是 UNTIL	这是 UNTIL
TYPEDEF		这是 TYPEDEF	这是 TYPEDEF
*=		这是 *=	这是 *=
/=		这是 /=	这是 /=
//		这是 //	这是 //

测试截图如下：



实验要求从词法分析上实现识别 ELSE，FOR，STEP，UNTIL，TYPEDEF *=，/= ，//。 对于操作符*=，/= ，//来说，我们只需在 GetSym 函数中对字符流进行判断，代码如下：

```
else if(CH=='*'){
        GetCh();
        if(CH=='='){
            Form1->printfs("这是*=");
            GetCh();
        } else{
            SYM=TIMES;
```

```
            }
    }else if(CH=='/'){
        GetCh();
        if(CH=='='){
            Form1->printfs("这是/=");
            GetCh();
        } else if(CH=='/'){
            Form1->printfs("这是//");
            GetCh();
        }
    }
```

对于识别保留字以及标识符来说，我们必须要在保留子表里添加相应的保留字

```
typedef enum  { NUL, IDENT, NUMBER, PLUS, MINUS, TIMES,
                SLASH, ODDSYM, EQL, NEQ, LSS, LEQ, GTR, GEQ,
                LPAREN, RPAREN, COMMA, SEMICOLON, PERIOD,
                BECOMES, BEGINSYM, ENDSYM, IFSYM, THENSYM,
                WHILESYM, WRITESYM, READSYM, DOSYM, CALLSYM,
                CONSTSYM, VARSYM, PROCSYM, PROGSYM, ELSESYM,
                    FORSYM, STEPSYM, UNTILSYM, TYPEDEFSYM
        } SYMBOL;
char *SYMOUT[] = {"NUL", "IDENT", "NUMBER", "PLUS", "MINUS", "TIMES",
        "SLASH", "ODDSYM", "EQL", "NEQ", "LSS", "LEQ", "GTR", "GEQ",
        "LPAREN", "RPAREN", "COMMA", "SEMICOLON", "PERIOD",
        "BECOMES", "BEGINSYM", "ENDSYM", "IFSYM", "THENSYM",
        "WHILESYM", "WRITESYM", "READSYM", "DOSYM", "CALLSYM",
        "CONSTSYM", "VARSYM", "PROCSYM", "PROGSYM", "ELSESYM",
            "FORSYM", "STEPSYM", "UNTILSYM", "TYPEDEFSYM" };
```

另外我们还需要修改一个全局常量 NORW，该变量控制了保留字表的大小，另外我们还需要修改 TForm1::ButtonRunClick 函数，将识别的词与保留字表里的词对应起来

```
void __fastcall TForm1::ButtonRunClick(TObject *Sender) {
  for (CH=' '; CH<='^'; CH++) SSYM[CH]=NUL;
  strcpy(KWORD[ 1],"BEGIN");     strcpy(KWORD[ 2],"CALL");
  strcpy(KWORD[ 3],"CONST");     strcpy(KWORD[ 4],"DO");
  strcpy(KWORD[ 5],"ELSE");      strcpy(KWORD[ 6],"END");
  strcpy(KWORD[ 7],"FOR");       strcpy(KWORD[ 8],"IF");
  strcpy(KWORD[ 9],"ODD");       strcpy(KWORD[10],"PROCEDURE");
  strcpy(KWORD[11],"PROGRAM");   strcpy(KWORD[12],"READ");
  strcpy(KWORD[13],"STEP");      strcpy(KWORD[14],"THEN");
  strcpy(KWORD[15],"TYPEDEF");   strcpy(KWORD[16],"UNTIL");
  strcpy(KWORD[17],"VAR");       strcpy(KWORD[18],"WHILE");
  strcpy(KWORD[19],"WRITE");
  WSYM[ 1]=BEGINSYM;    WSYM[ 2]=CALLSYM;
  WSYM[ 3]=CONSTSYM;    WSYM[ 4]=DOSYM;
  WSYM[ 5]=ELSESYM;     WSYM[ 6]=ENDSYM;
  WSYM[ 7]=FORSYM;      WSYM[ 8]=IFSYM;
  WSYM[ 9]=ODDSYM;      WSYM[10]=PROCSYM;
  WSYM[11]=PROGSYM;     WSYM[12]=READSYM;
  WSYM[13]=STEPSYM;     WSYM[14]=THENSYM;
  WSYM[15]=TYPEDEFSYM;  WSYM[16]=UNTILSYM;
  WSYM[17]=VARSYM;      WSYM[18]=WHILESYM;
  WSYM[19]=WRITESYM;
  SSYM['+']=PLUS;       SSYM['-']=MINUS;
  SSYM['*']=TIMES;      SSYM['/']=SLASH;
  SSYM['(']=LPAREN;     SSYM[')']=RPAREN;
  SSYM['=']=EQL;        SSYM[',']=COMMA;
  SSYM['.']=PERIOD;
  SSYM[';']=SEMICOLON;
  strcpy(MNEMONIC[LIT],"LIT");    strcpy(MNEMONIC[OPR],"OPR");
  strcpy(MNEMONIC[LOD],"LOD");    strcpy(MNEMONIC[STO],"STO");
  strcpy(MNEMONIC[CAL],"CAL");    strcpy(MNEMONIC[INI],"INI");
  strcpy(MNEMONIC[JMP],"JMP");    strcpy(MNEMONIC[JPC],"JPC");
```

另外要实现识别保留字的功能还需要在 statement 函数中保留新添加的保留字的 case 语句

```
case FORSYM:
        GetSym();
        Form1->printfs("这是 FOR");
        break;
    case STEPSYM:
        GetSym();
        Form1->printfs("这是 STEP");
        break;
    case UNTILSYM:
        GetSym();
```

```
                Form1->printfs("这是  UNTIL");
                break;
        case TYPEDEFSYM:
                GetSym();
                Form1->printfs("这是  TYPEDEF");
                break;
```

4.1.3 实验 3

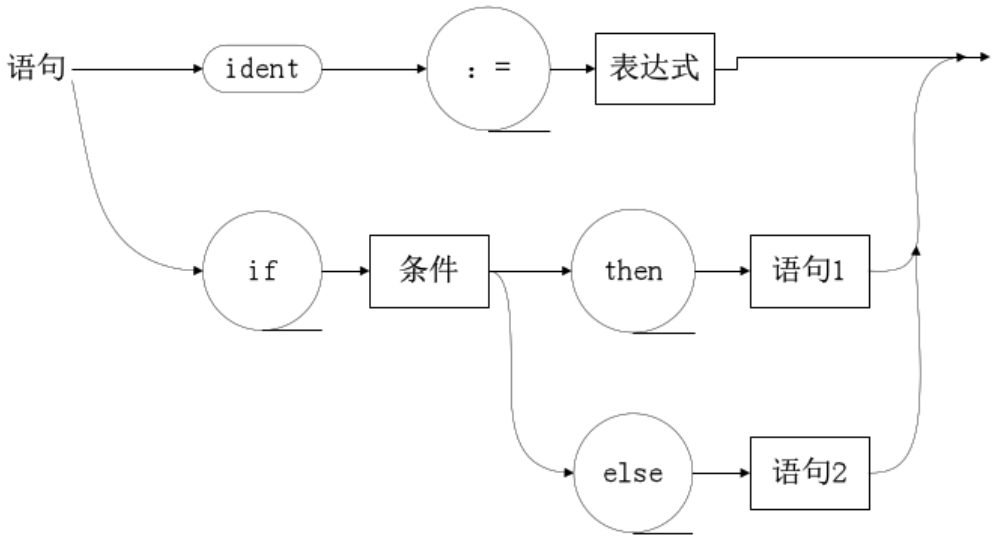
增加条件语句的 ELSE 子句（实现语法语义目标代码），

要求：写出相关文法和语法图，分析语义规则的实现。

文法：

G(S):S –> IF S ELSE S | IF S |A

语法图:



测试代码：

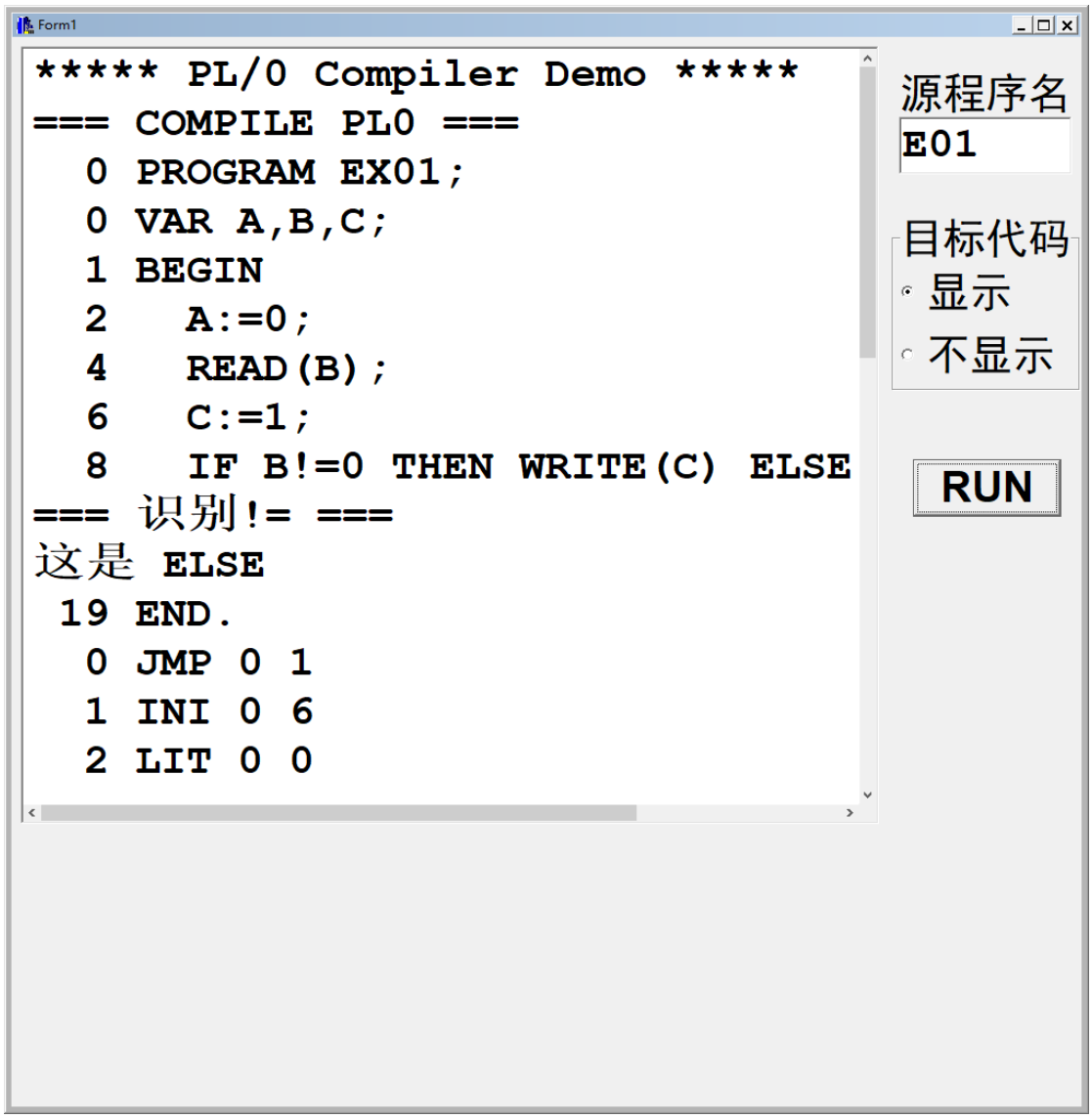
PROGRAM EX01;

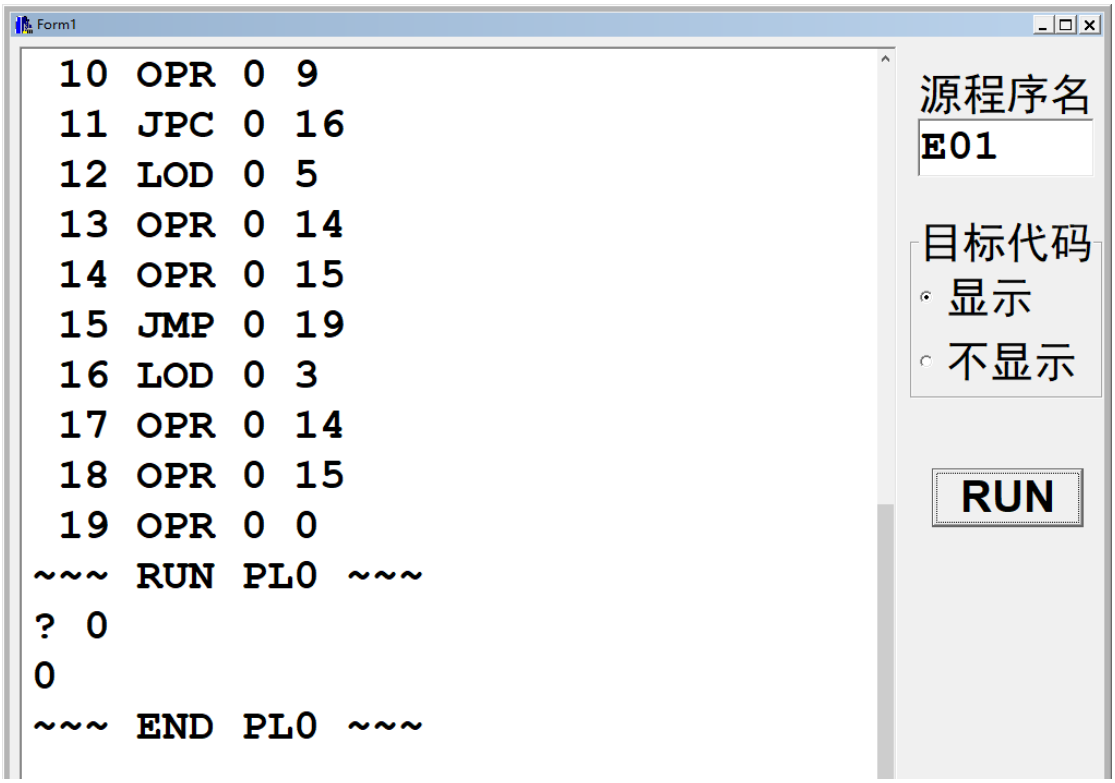
VAR A,B,C;

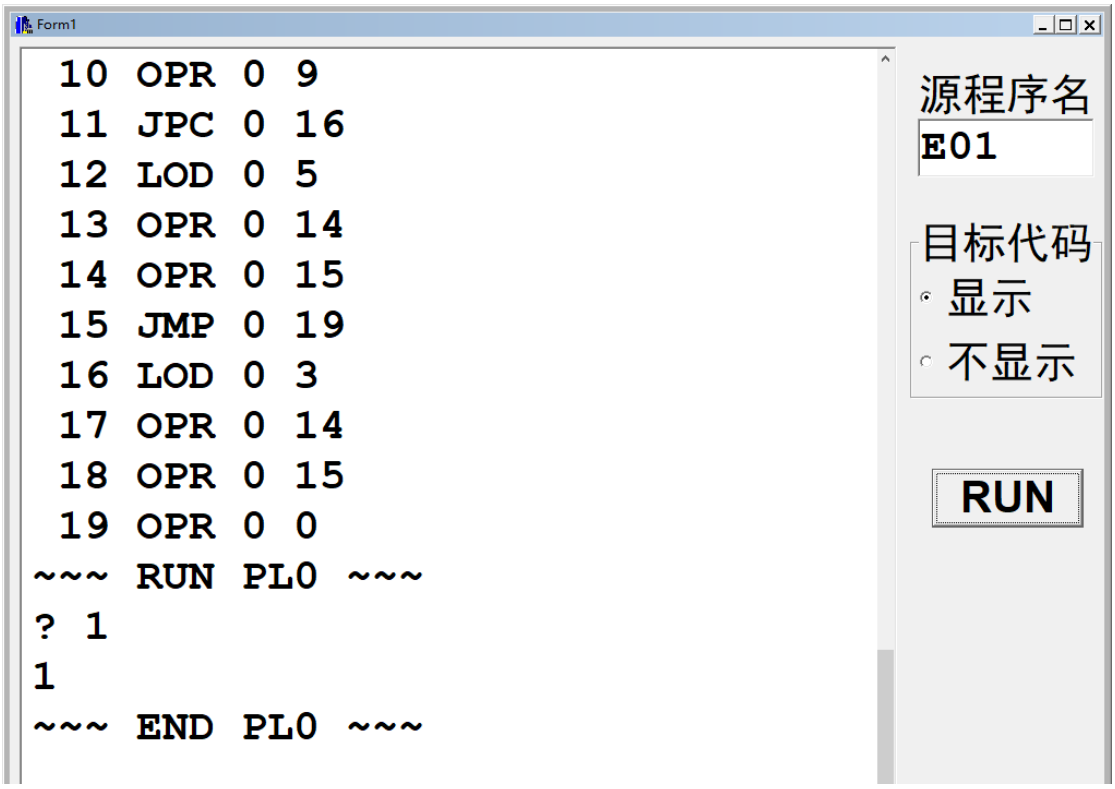
BEGIN

```
   A:=0;
   READ(B);
   C:=1;
   IF B!=0 THEN WRITE(C) ELSE WRITE(A);
END.
```

功能描述	测试 ELSE	
用例目的	测试 ELSE 关键字	
前提条件	无	
输入/动作	期望输出	实际情况
B=0	输出 1	输出 1
B=1	输出 0	输出 0


Form1
***** PL/0 Compiler Demo *****
=== COMPILE PL0 ===
0 PROGRAM EX01;
0 VAR A,B,C;
1 BEGIN
2 A:=0;
4 READ(B);
6 C:=1;
8 IF B!=0 THEN WRITE(C) ELSE
=== 识别!= ===
这是 ELSE
19 END.
0 JMP 0 1
1 INI 0 6
2 LIT 0 0
源程序名
E01
目标代码
显示
不显示
RUN





实现 3 要实现 ELSE 子句的语法功能。在先前识别 ELSE 的基础上我们还需要修改 statement。注意到 ELSE 子句并不能独立存在，因此我们把 ELSE 子句添加到 IF 的 case 块中。

```
case IFSYM:
    GetSym();
    CONDITION(SymSetUnion(SymSetNew(THENSYM,DOSYM),FSYS),LEV,TX);
    if (SYM==THENSYM)
        GetSym();
    else
        Error(16);
    CX1=CX;
    GEN(JPC,0,0);
    STATEMENT(SymSetUnion(SymSetNew(ELSESYM),FSYS),LEV,TX);
    if(SYM!=ELSESYM)
        CODE[CX1].A=CX;
    else{
        GetSym();
        Form1->printfs("这是 ELSE");
        CX2=CX;
        GEN(JMP,0,0);
        CODE[CX1].A=CX;
        STATEMENT(FSYS,LEV,TX);
        CODE[CX2].A=CX;
    }
    break;
```

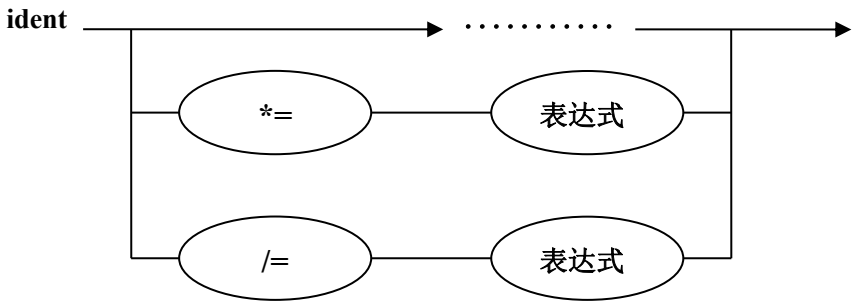
4.2 选做

4.2.1 实验 1

（1）首先修改关键字数目：

```
//--------------------------------------------------------------------------
const  AL    =  10;  //符号（标识符）的最大长度
const  NORW  =  21;  //关键字个数
const  TXMAX = 100;  //名字表容量
const  NMAX  =  14;  //整型数的最大长度
const  LMAX  =8;     //实型精确到小数点后的位数
const  AMAX  =2047;  //栈地址上界
const  LEVMAX=   3;  //嵌套最大层数
const  CXMAX = 200;  //虚拟机代码数
```

（2）语法描述图如下：



（3）修改部分如下：

```
SYMSET SymSetUnion(SYMSET S1, SYMSET S2) {
  SYMSET S=(SYMSET)malloc(sizeof(int)*47);
  for (int i=0; i<47; i++)
        if (S1[i] || S2[i]) S[i]=1;
        else S[i]=0;
  return S;
}
```

```
SYMSET SymSetAdd(SYMBOL SY, SYMSET S) {
  SYMSET S1;
  S1=(SYMSET)malloc(sizeof(int)*47);
  for (int i=0; i<47; i++) S1[i]=S[i];
  S1[SY]=1;
  return S1;
}
```

```
//-------------------------------------------------------
SYMSET SymSetNew(SYMBOL a) {
  SYMSET S; int i,k;
  S=(SYMSET)malloc(sizeof(int)*47);
  for (i=0; i<47; i++) S[i]=0;
  S[a]=1;
  return S;
}
```

```
//-------------------------------------------------------------------
SYMSET SymSetNew(SYMBOL a, SYMBOL b) {
  SYMSET S; int i,k;
  S=(SYMSET)malloc(sizeof(int)*47);
  for (i=0; i<47; i++) S[i]=0;
  S[a]=1;  S[b]=1;
  return S;
}
```

```
//-----------------------------------------------------------
SYMSET SymSetNew(SYMBOL a, SYMBOL b, SYMBOL c) {
  SYMSET S; int i,k;
  S=(SYMSET)malloc(sizeof(int)*47);
  for (i=0; i<47; i++) S[i]=0;
  S[a]=1;  S[b]=1; S[c]=1;
  return S;
}
```

```
SYMSET SymSetNew(SYMBOL a, SYMBOL b, SYMBOL c, SYMBOL d) {
  SYMSET S; int i,k;
  S=(SYMSET)malloc(sizeof(int)*47);
  for (i=0; i<47; i++) S[i]=0;
  S[a]=1;  S[b]=1; S[c]=1; S[d]=1;
  return S;
}
```

```
//-----------------------------------------------------------------------------
SYMSET SymSetNew(SYMBOL a, SYMBOL b, SYMBOL c, SYMBOL d,SYMBOL e, SYMBOL f) {
  SYMSET S; int i,k;
  S=(SYMSET)malloc(sizeof(int)*47);
  for (i=0; i<47; i++) S[i]=0;
  S[a]=1;  S[b]=1; S[c]=1; S[d]=1; S[e]=1; S[f]=1;
  return S;
}
```

```
//-------------------------------------------------------------
SYMSET SymSetNULL() {
  SYMSET S; int i,n,k;
  S=(SYMSET)malloc(sizeof(int)*47);
  for (i=0; i<47; i++) S[i]=0;
  return S;
}
```

```
    DECLBEGSYS=(int*)malloc(sizeof(int)*47);
    STATBEGSYS=(int*)malloc(sizeof(int)*47);
    FACBEGSYS =(int*)malloc(sizeof(int)*47);
```

```
  } else if(CH=='*') {
        GetCh();
        if(CH=='=') { SYM=TIMESBECOMES; GetCh(); }
        else SYM=TIMES;
  } else if(CH=='/') {
        GetCh();
        if(CH=='=') { SYM=SLASHBECOMES; GetCh(); }
        else if(CH=='*') {
                GetCh();
                i=CH;
                while(i!='*' || CH!='/') {
                        i=CH;
                        GetCh();
                       // Form1->printcs(CH);
                }
                if(i!='*'&&CH!='/') Error(19);
                else {
                        GetCh();
                        GetSym();
                }
        } else SYM=SLASH;
```

（4）运行示例及结果展示：

```
=== COMPILE PL0 ===
  0 PROGRAM EX01;
  0 VAR A,B,C,D;
  1 BEGIN
  2     A:=15;
  4     B:=75;
  6     C:=5;
  8     D:=3;
 10     B/=D;
 14     WRITE(B);
 17     D*=A;
 21     WRITE(D);
 24 END.
  0 JMP 0 1
```

```
  0 JMP 0 1
  1 INI 0 7
  2 LIT 0 15
  3 STO 0 3
  4 LIT 0 75
  5 STO 0 4
  6 LIT 0 5
  7 STO 0 5
  8 LIT 0 3
  9 STO 0 6
 10 LOD 0 4
 11 LOD 0 6
 12 OPR 0 5
 13 STO 0 4
```

```
 15 OPR 0 14
 16 OPR 0 15
 17 LOD 0 6
 18 LOD 0 3
 19 OPR 0 4
 20 STO 0 6
 21 LOD 0 6
 22 OPR 0 14
 23 OPR 0 15
 24 OPR 0 0
~~~ RUN PL0 ~~~
25
45
~~~ END PL0 ~~~
```

4.2.2 实验 2

（1）格式如下：

FOR <变量>:=<表达式> STEP <表达式>UNTIL <表达式> DO <语句>

（2）语法描述图：

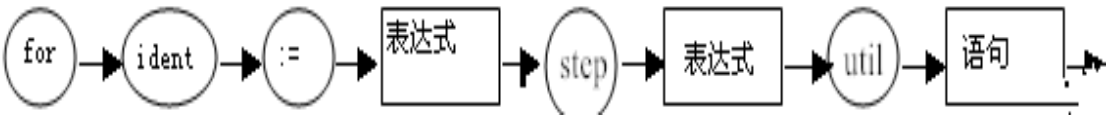


图 4.16 语法描述图

（3）修改部分如下：

词法分析部分增加关键字：

```
//---------------------------------------------------------------------------
void __fastcall TForm1::ButtonRunClick(TObject *Sender) {
  for (CH=' '; CH<='^'; CH++) SSYM[CH]=NUL;
  strcpy(KWORD[ 1],"BEGIN");     strcpy(KWORD[ 2],"CALL");
  strcpy(KWORD[ 3],"CHAR");
  strcpy(KWORD[ 4],"CONST");     strcpy(KWORD[ 5],"DO");
  strcpy(KWORD[ 6],"ELSE");
  strcpy(KWORD[ 7],"END");     strcpy(KWORD[ 8],"FLOAT");
  strcpy(KWORD[ 9],"FOR");
  strcpy(KWORD[10],"IF");
  strcpy(KWORD[11],"ODD");       strcpy(KWORD[12],"PROCEDURE");
  strcpy(KWORD[13],"PROGRAM");   strcpy(KWORD[14],"READ");
  strcpy(KWORD[15],"RETURN");
  strcpy(KWORD[16],"STEP");
  strcpy(KWORD[17],"THEN");
  strcpy(KWORD[18],"UNTIL");
  strcpy(KWORD[19],"VAR");
  strcpy(KWORD[20],"WHILE");     strcpy(KWORD[21],"WRITE");
  WSYM[ 1]=BEGINSYM;    WSYM[ 2]=CALLSYM;
```

```
//扩充 for 语句
case FORSYM:
        GetSym();
        STATEMENT(SymSetUnion(SymSetNew(STEPSYM),FSYS),LEV,TX);
        CX3=CX; GEN(JMP,0,0);
        CX1=CX;
        //识别 Step 关键字
        if(SYM==STEPSYM) {
                GetSym();

STATEMENT(SymSetUnion(SymSetNew(UNTILSYM),FSYS),LEV,TX);
        } else Error(8);
        //识别 util 关键字
        if(SYM==UNTILSYM) {
                CODE[CX3].A=CX;
                GetSym();
        } else Error(8);
        CONDITION(SymSetAdd(DOSYM, FSYS),LEV,TX);
        //识别 do 关键字
        if(SYM==DOSYM) {
                GetSym();
        } else Error(8);
        CX2=CX;    GEN(JPC,0,0);
        STATEMENT(FSYS,LEV,TX);
        GEN(JMP,0,CX1);
        CODE[CX2].A=CX;
        break;
```

```
strcpy(KWORD[19],"VAR");
strcpy(KWORD[20],"WHILE");     strcpy(KWORD[21],"WRITE
WSYM[ 1]=BEGINSYM;    WSYM[ 2]=CALLSYM;
WSYM[ 3]=CHARSYM;
WSYM[ 4]=CONSTSYM;
WSYM[ 5]=DOSYM;
WSYM[ 6]=ELSESYM;     WSYM[ 7]=ENDSYM;
WSYM[ 8]=FLOATSYM;
WSYM[ 9]=FORSYM;      WSYM[ 10]=IFSYM;
WSYM[11]=ODDSYM;      WSYM[12]=PROCSYM;
WSYM[13]=PROGSYM;     WSYM[14]=READSYM;
WSYM[15]=RETURNSYM;   WSYM[16]=STEPSYM;
WSYM[17]=THENSYM;
WSYM[18]=UNTILSYM;    WSYM[19]=VARSYM;
WSYM[20]=WHILESYM;    WSYM[21]=WRITESYM;
```

（3）运行示例程序及结果：

```
***** PL/0 Compiler Demo *****
=== COMPILE PL0 ===
  0 PROGRAM EX01;
  0 VAR I,B,SUM;
  1 BEGIN
  2     I:=0;
  4     B:=20;
  6    FOR I:=1 STEP I+=1 UNTIL
 16     SUM+=B;
 22    WRITE(SUM);
 25 END.
  0 JMP 0 1
  1 INI 0 6
  2 LIT 0 0
```

```
 15 OPR 0 13
 16 JPC 0 22
 17 LOD 0 5
 18 LOD 0 4
 19 OPR 0 2
 20 STO 0 5
 21 JMP 0 9
 22 LOD 0 5
 23 OPR 0 14
 24 OPR 0 15
 25 OPR 0 0
~~~ RUN PL0 ~~~
400
~~~ END PL0 ~~~
```

5 开发过程和完成情况

5.1 开发过程

经过这个学期的编译原理课程的学习，发现这么课程跟之前的很多课程相比，难度大很多，里面很多东西不容易理解，通常一头雾水，比如语法分析中讲规约过程，刚开始读会很不理解为什么这么规约，直到读完后面的内容。

在开发的过程中，通过数天的不间断的完成，我了解了编译的一整个过程，对于 PL0 的程序书写也遇到了许多困难，也是通过与同学讨论以及翻阅课本以及查阅资料来完成的。实现内容中基本内容统统实现，对于一些错误处理，也能够在 debug 的次数多了后能够熟练的掌握出错编号所对应的出错原因。总之收获不少。

5.2 完成情况

在经过理论上的学习后，我也积极参与实践中，本学期编译原理的必做的 3 个实验的实验要求，我已全部完成，并且完成了选做中的两个实验。

实验过后，使我对编译原理这么课程有了更深刻的解了，把死板的课本知识变得生动有趣，激发了学习的积极性，提升了自己的动手能力，加深了对理论知识的理解，受益良多！