廣東工業大學

实验报告

课程名称 编译原理

题目名称 PL/0 编译程序的修改扩充

学生学院 计算机学院

专业班级 软件工程 1 班

学　　号 3119005028

学生姓名 魏耀辉

指导教师 张　巍

2022 年 06 月 23 日

目录

一、概述

本学期编译原理实验要求对基础的 PL/0 编译程序进行修改扩充，要求使用标准的 PL/0 语言进行修改扩充，本人使用 C++Builder 软件进行对 PL/0 源程序进行修改扩充，并最终完成该实验。

二、实验内容与要求

（一）课内实验

对 PL/0 作以下修改扩充：

（1）增加单词：保留字 ELSE,FOR,STEP,UNTIL,DO,RETURN 以及运算符*=，/=，&，|，！

（2）修改单词：不等号#修改为<>

（3）增加条件语句的 ELSE 子句，要求写出相关文法，语法描述图，语意描述图

（二）选做内容

对 PL/0 继续作以下修改扩充：

（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<表达式>

（3）增加类型：①字符类型②实数类型

（4）扩充运算：++和--（要求作为表达式实现）

（5）扩充函数：①有返回值和返回语句②有参数函数

（6）增加一维数组类型（可增加指令）

（7）其他典型语言设施

实验要求使用老师给定的 PL/0 基础源程序进行修改扩充，使扩充后的 PL/0 程序完成全部的课内实验，并鼓励完成选做内容

三、实验环境与工具

源语言：PL/0 语言，后缀名为.PL0

目标语言：.COD 的目标代码

实现平台：Borland C++ Builder 6

运行平台：Windows10

四、实验实现内容

（一）课内实验实现内容

（1）增加单词：保留字 ELSE,FOR,STEP,UNTIL,DO,RETURN 以及运算符*=，/=，&，|，！

（2）修改单词：不等号#修改为<>

（3）增加条件语句的 ELSE 子句，要求写出相关文法，语法描述图，语意描述图

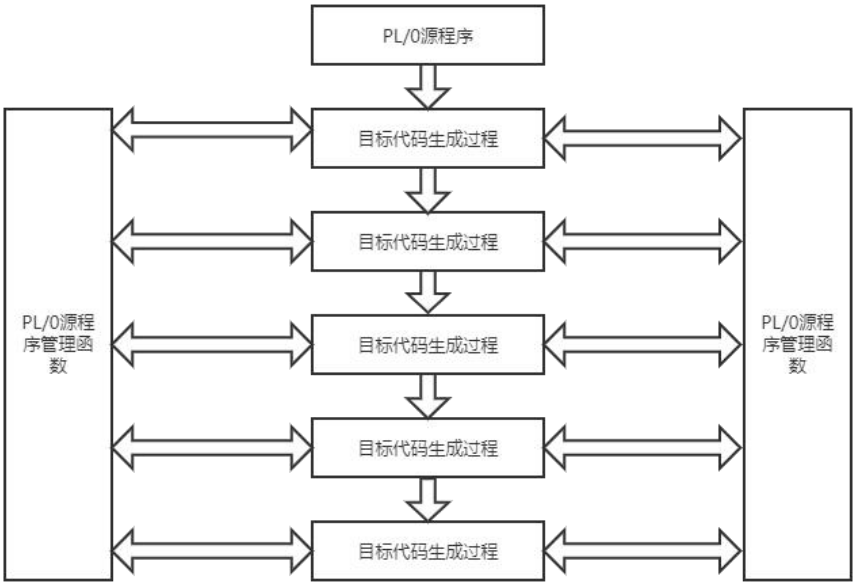
（二）选做部分实现内容

（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<表达式>

五、结构设计说明

（一）PL/0 编译程序的结构图



（二）各功能模块描述

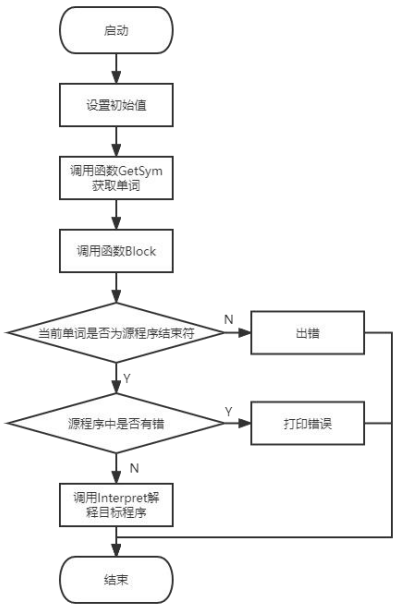
各功能模块描述如下表 5.1 所示：

表 5.1 各功能模块描述

函数名及其参数	功能描述
Error(int n)	出错处理，打印程序出错位置和编码
Getch()	读取一个字符
GetSym()	词法分析，读取一个单词
GEN(FCT X,int Y,int Z)	生成目标代码，并送入目标程序区
TEST(SYMSET S1,SYMSET S2,int N)	测试当前单词符号时候合法
ENTER(OBJECTS K,int LEV,int &TX,int &DX)	名字表中添加一项
POSITION(ALFA ID,int TX)	查找标识符在名字表中的位置
ConstDeclaration(int LEV,int &TX,int &DX)	常量声明处理
VarDeclaration(int LEV,int &TX,int &DX)	变量声明处理
ListCode(int CX0)	输出目标代码清单
FACTOR(SYMSET FSYS,int LEV,int &TX)	因子处理
TERM(SYMSET FSYS,int LEV,int &TX)	项处理
EXPRESSION(SYMSET FSYS,int LEV,int &TX)	表达式处理
CONDITION(SYMSET FSYS,int LEV,int &TX)	条件处理
STATEMENT(SYMSET FSYS,int LEV,int &TX)	语句处理
Block(int LEV,int TX,SYMSET FSYS)	分程序分析处理过程
BASE(int L,int B,int S[])	通过程序基址求上一层过程基址
Interpret()	对目标代码的解析执行程序

（三）PL/0 编译程序流程图

PL/0 编译程序流程图如下图所示：



六、主要成分描述

（一）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。这些值是由编译程序本身联系到相应标识符上去的。这种联系实在处理常熟、变量和过程说明完成的。为此，符号表应该包含每个标识符所联系的属性，根据符号表中的属性可以分为 CONSTANT 常量、VARIABLE 变量、PROCEDUR 过程。编译的任务就是确定存放该值的地址。符号表的功能有：确定符号的属性、上下文语义合法性检查的依据、目标代码生成阶段地址分配的依据。本次实验中并未改变符号表的定义，符号表定义如下表 6.1 所示：

表 6.1 符号表定义

```
typedef    char ALFA[11];//符号名
typedef    enum { CONSTANT, VARIABLE, PROCEDUR } OBJECTS ;//符号属性
struct {
   ALFA NAME;
   OBJECTS KIND;
   union {
      int VAL;      /*CONSTANT*/
      struct { int LEVEL,ADR,SIZE; } vp;    /*VARIABLE,PROCEDUR:*/
   };
} TABLE[TXMAX];//符号表的数据结构
```

（二）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经常使用。

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，指令指针会指到 0 位置，把这样一个条件作为终止循环的条件，保证程序可以正常的结束。

本次实验中，并未更改运行栈的定义。

（三）语法语义分析方法

语法分析的任务是识别由词法分析给出的单词符号序列在结构上是否符合给定的文法规则。PL/0 编译程序的语法分析采用了自顶向下的递归子程序法。对于每个非终结符语法单元，编一个独立的处理过程（子程序）。语法分析从读入第一个单词开始由开始符[程序]出发，沿语法描述图箭头所指方向进行分析。当遇到非终结符时，则调用相对应的处理过程，从语法描述图也就进入了一个语法单元，再沿当前所进入的语法描述图的箭头方向进行分析，当遇到描述图中是终结符时，则判断当前读入的单词是否与图中的终结符相匹配，若匹配，则执行对应的语义程序（翻译程序）。再读入下一个单词继续分析。遇到分支点时将当前的单词与分支点上的多个终结符逐个比较，若都不匹配时可能是进入下一非终结符语法单位或是出错。

语法分析主要由分程序分析过程（BLOCK）常量定义分析过程（ConstDeclaration）、变量定义分析过程（VarDeclaration）、字符变量定义分析过程（CharDeclaration）、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。

分程序语法描述图如图 6.1 所示：

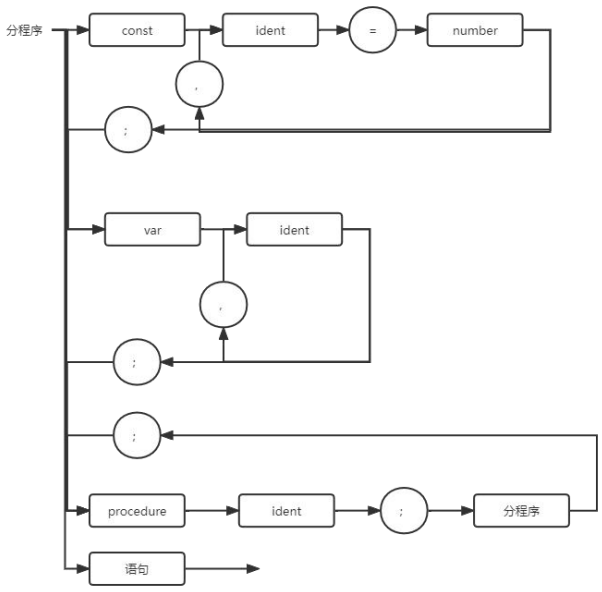


图 6.1 分程序语法描述图

语句语法描述图如图 6.2 所示：

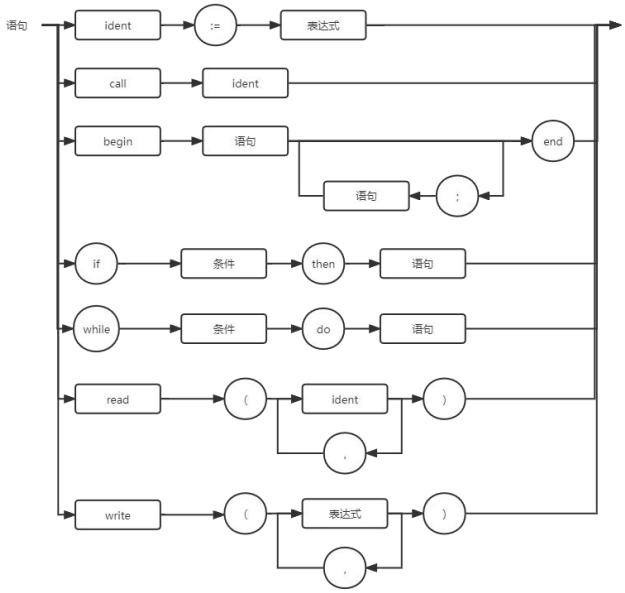


图 6.2 语句语法描述图

条件语法描述图如图 6.3 所示：

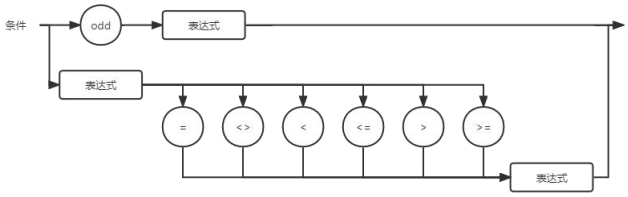


图 6.3 条件语法描述图

表达式语法描述图如图 6.4 所示：

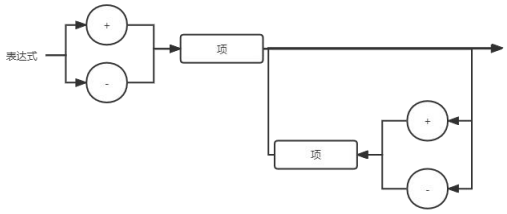


图 6.4 表达式语法描述图

项语法描述图如图 6.5 所示：

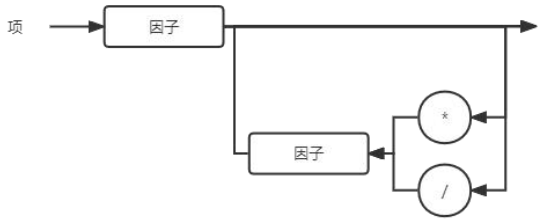


图 6.5 项语法描述图

因子语法描述图如图 6.6 所示：

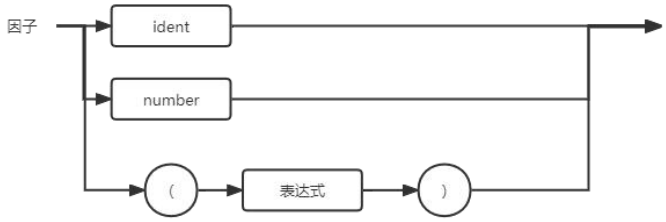


图 6.6 因子语法描述图

语义描述图如图 6.7 所示：

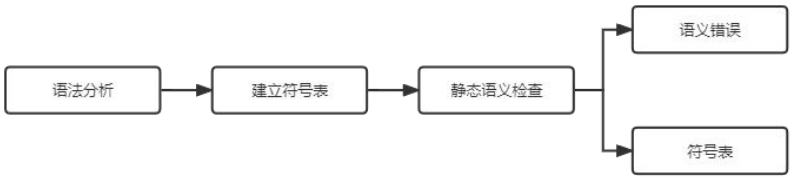


图 6.7 语义描述图

（四）代码生成

PL/0 编译程序所产生的目标代码是一种假想栈式计算机的汇编语言，可称为类 PCODE 指令，代码，它不依赖于任何实际计算机，其指令集十分简单，详细指令集如下表 6.2 所示：

表 6.2 指令表

指令	说明
CAL L A	调用地址为 A 的过程（置指令地址寄存器为 A）；L 为调用过程与被调用过程的层差；设置被调用过程的 3 个联系单元
INT 0 A	在栈顶开辟 A 个存储单元，服务于被调用的过程;A 等于该过程的局部变量数加 3;3 个特殊的联系单元
JMP 0 A	无条件转移至地址 A，即置指令地址寄存器为 A
JPC 0 A	条件转移指令；若栈顶为 0，则转移至地址 A，即置指令地址寄存器为 A ；T 减 1
LIT 0 A	立即数存入栈顶，即置 T 所指存储单元的值为 A1； T 加 1
LOD L A	将层差为 L、偏移量为 A 的存储单元的值取到栈顶； T 加 1
OPR 0 0	过程调用结束后,返回调用点并退栈； 重置基址寄存器和栈顶寄存器
OPR 0 1	求栈顶元素的相反数，结果值留在栈顶
OPR 0 2	次栈顶与栈顶的值相加，结果存入次栈顶； T 减 1
OPR 0 3	次栈顶的值减去栈顶的值，结果存入次栈顶； T 减 1
OPR 0 4	次栈顶的值乘以栈顶的值，结果存入次栈顶； T 减 1
OPR 0 5	次栈顶的值除以栈顶的值，结果存入次栈顶； T 减 1
OPR 0 6	栈顶元素的奇偶判断，若为奇数，结果为 1；若为偶数，结果为 0 ；结果值留在栈顶
OPR 0 8	比较次栈顶与栈顶是否相等；若相等，结果为 0；存结果至次栈顶；T 减 1
OPR 0 9	比较次栈顶与栈顶是否不相等；若不相等，结果为 0；存结果至次栈顶；T 减 1
OPR 0 10	比较次栈顶是否小于栈顶； 若小于，结果为 0；存结果至次栈顶；T 减 1
OPR 0 11	比较次栈顶是否大于等于栈顶； 若大于等于，结果为 0；存结果至次栈顶；T 减 1
OPR 0 12	比较次栈顶是否大于栈顶； 若大于，结果为 0；存结果至次栈顶；T 减 1
OPR 0 13	比较次栈顶是否小于等于栈顶； 若小于等于，结果为 0；存结果至次栈顶；T 减 1
OPR 0 14	栈顶的值输出至控制台屏幕； T 减 1
OPR 0 15	控制台屏幕输出一个换行
OPR 0 16	从控制台读入一行输入，置入栈顶； T 加 1
STO L A	T 减 1； 将栈顶的值存入层差为 L、偏移量为 A 的存储单元

本次实验中并未对指令进行修改或扩充。

七、测试用例

（一）测试用例 1

测试用例 1 用于测试 DO、RETURN 关键字以及*=，/=，&，|，！等运算符

```
测试用例 1：TEST1.PL0
PROGRAM TEST1;
BEGIN
   DO;
   UNTIL;
   RETURN;
   *=;
   /=;
   &;
   |;
   !;
END.
```

（二）测试用例 2

测试用例 2 用于测试 ELSE 关键字、不等号<>以及增加条件语句的 ELSE 子句

```
测试用例 2：TEST2.PL0
PROGRAM TEST2;
VAR A,B,C;
BEGIN
  A:=6;
  B:=4;
  C:=2;
  IF A<>6 THEN
    WRITE(B)
  ELSE
    WRITE(C);
  A:=4;
  IF A<>6 THEN
    WRITE(B)
  ELSE
    WRITE(C);
END.
```

（三）测试用例 3

测试用例 3 用于测试*=和/=的赋值运算操作

```
测试用例 3：TEST3.PL0
PROGRAM TEST3;
VAR A,B,C,D;
BEGIN
    A:=10;
    B:=2;
    C:=9;
    D:=3;
    A*=B;
    C/=D;
    WRITE(A);
    WRITE(C);
END.
```

（四）测试用例 4

测试用例 4 用于测试 FOR 语句的扩充以及 FOR、STEP、UNTIL 等关键字

```
测试用例 4：TEST4.PL0
PROGRAM TEST4;
VAR A,B;
BEGIN
B:=0;
FOR A:=0 STEP 1 UNTIL 9
     DO
    BEGIN
    B:=B+1;
    END;
;
WRITE(A);
WRITE(B);
END.
```

八、开发过程及完成情况

（一）增加单词：保留字 ELSE,FOR,STEP,UNTIL,DO,RETURN 以及运算符*=，/=，&，|，！修改单词

（1）参照已有保留字，将新增保留字添加到对应的保留字集合中

（2）将新增的保留字按照字母表升序的方式添加，注意需要符号与 SYM 的对应

（3）在 GetSym()函数中在相应位置增加相应的运算符分析判断

（4）在 Statement()函数中增加相应的语句，用于检验保留字是否添加成功

具体操作如下：

（1）

```
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,
                //增加 ELSE,FOR,STEP,UNTIL,RETURN
                ELSESYM, FORSYM, STEPSYM, UNTILSYM, RETURNSYM,
                //增加*=,/=,&,||,!
                TIMESBECOMES, SLASHBECOMES, ANDSYM, ORSYM, NOTSYM
        } 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",
        //新增 ELSE,FOR,STEP,UNTIL,RETURN
        "ELSESYM", "FORSYM", "STEPSYM", "UNTILSYM", "RETURNSYM",
        //新增*=,/=,&,||,!
        "TIMESBECOMES", "SLASHBECOMES", "ANDSYM", "ORSYM", "NOTSYM"
        };
```

（2）

```
 strcpy(KWORD[ 5],"ELSE");      // 增加保留字 1，ELSE
 strcpy(KWORD[ 7],"FOR");       // 增加保留字 2，FOR
 strcpy(KWORD[13],"RETURN");    // 增加保留字 5，RETURN
 strcpy(KWORD[14],"STEP");      // 增加保留字 3，STEP
 strcpy(KWORD[16],"UNTIL");     // 增加保留字 4，UNTIL
 WSYM[ 5]=ELSESYM;      // 增加保留字符号 1，ELSESYM
 WSYM[ 7]=FORSYM;       // 增加保留字符号 2，FORSYM
 WSYM[13]=RETURNSYM;    // 增加保留字符号 5，RETURNSYM
 WSYM[14]=STEPSYM;      // 增加保留字符号 3，STEPSYM
 WSYM[16]=UNTILSYM;     // 增加保留字符号 4，UNTILSYM
 SSYM['&']=ANDSYM;      // 新增符号，&
```

```
  SSYM['!']=NOTSYM;     // 新增符号，!
（3）
void GetSym() {
  int i,J,K;   ALFA  A;
  while (CH<=' ') GetCh();
  if (CH>='A' && CH<='Z') { /*ID OR RESERVED WORD*/
    K=0;
    do {
      if (K<AL) A[K++]=CH;
      GetCh();
    }while((CH>='A' && CH<='Z')||(CH>='0' && CH<='9'));
    A[K]='\0';
    strcpy(ID,A); i=1; J=NORW;
    do {
      K=(i+J) / 2;
      if (strcmp(ID,KWORD[K])<=0) J=K-1;
      if (strcmp(ID,KWORD[K])>=0) i=K+1;
    }while(i<=J);
    if (i-1 > J) SYM=WSYM[K];
    else SYM=IDENT;
  }
  else
    if (CH>='0' && CH<='9') { /*NUMBER*/
      K=0; NUM=0; SYM=NUMBER;
      do {
        NUM=10*NUM+(CH-'0');
        K++; GetCh();
      }while(CH>='0' && CH<='9');
      if (K>NMAX) Error(30);
    }
    else
      if (CH==':') {
        GetCh();
        if (CH=='=') { SYM=BECOMES; GetCh(); }
        else SYM=NUL;
      }
      else /* THE FOLLOWING TWO CHECK WERE ADDED
              BECAUSE ASCII DOES NOT HAVE A SINGLE CHARACTER FOR <= OR >= */
        if (CH=='<') {
          GetCh();
          if (CH=='=') { SYM=LEQ; GetCh(); }
          //新增不等号运算符
          else if (CH=='>') { SYM=NEQ;GetCh();}
          else SYM=LSS;
```

```
        }
        else
          if (CH=='>') {
            GetCh();
            if (CH=='=') { SYM=GEQ; GetCh(); }
            else SYM=GTR;
          }
    //新增*=,/=,&,||,!运算符
    else if(CH=='*') {      // *=
        GetCh();
        if(CH=='=') { SYM=TIMESBECOMES; GetCh(); }
        else SYM=TIMES;
    } else if(CH=='/') {    // /=
        GetCh();
        if(CH=='=') { SYM=SLASHBECOMES; GetCh(); }
        else SYM=SLASH;
    } else if(CH=='&') {    // &
        GetCh();
        SYM=ANDSYM;
    } else if(CH=='|') {    // ||
        SYM=ORSYM;
        GetCh();
    } else if(CH=='!') {    // !
        GetCh();
        SYM=NOTSYM;
    }
    else { SYM=SSYM[CH]; GetCh(); }
} /*GetSym()*/
```

（4）

```
// 用来检验保留字是否添加成功的标志
    case FORSYM:
        GetSym();
        Form1->printfs("识别保留字 FORSYM 成功");
        //GetSym();
        if (SYM != IDENT)
            Error(47);
        else
            i = POSITION(ID, TX);
        if (i == 0)
            Error(11);
        else if (TABLE[i].KIND != VARIABLE)
        {
            Error(12);
            i = 0;
```

```
        }
        GetSym();
        if (SYM == BECOMES)
            GetSym();
        else
            Error(13);
        EXPRESSION(SymSetUnion(SymSetNew(STEPSYM), FSYS), LEV, TX);
        if (i != 0)
            GEN(STO, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        if (SYM == STEPSYM)
            GetSym();
        else
            Error(46);
        CX1 = CX;
        GEN(JMP, 0, 0);
        CX3 = CX;
        EXPRESSION(SymSetUnion(SymSetNew(UNTILSYM), FSYS), LEV, TX);
        GEN(LOD, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        GEN(OPR, 0, 2);
        GEN(STO, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        if (SYM == UNTILSYM)
            GetSym();
        else
            Error(47);
        CODE[CX1].A = CX;
        EXPRESSION(SymSetUnion(SymSetNew(DOSYM), FSYS), LEV, TX);
        GEN(LOD, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        GEN(OPR, 0, 11);
        CX2 = CX;
        GEN(JPC, 0, 0);
        if (SYM == DOSYM)
            GetSym();
        else
            Error(48);
        STATEMENT(FSYS,LEV,TX);
        GEN(JMP, 0, CX3);
        CODE[CX2].A = CX;
        break;
    case STEPSYM:
        GetSym();
        Form1->printfs("识别保留字 STEPSYM 成功");
        break;
    case UNTILSYM:
        GetSym();
```

```
        Form1->printfs("识别保留字 UNTILSYM 成功");
        break;
    case RETURNSYM:
        GetSym();
        Form1->printfs("识别保留字 RETURNSYM 成功");
        break;
    case DOSYM:
        GetSym();
        Form1->printfs("识别保留字 DOSYM 成功");
        break;
    // 用来检验运算符是否添加成功的标志。
    case TIMESBECOMES:
        GetSym();
        Form1->printfs("识别运算符*=成功");
        break;
    case SLASHBECOMES:
        GetSym();
        Form1->printfs("识别运算符/=成功");
        break;
    case ANDSYM:
        GetSym();
        Form1->printfs("识别运算符&成功");
        break;
    case ORSYM:
        GetSym();
        Form1->printfs("识别运算符|成功");
        break;
    case NOTSYM:
        GetSym();
        Form1->printfs("识别运算符!成功");
        break;
  }
```

（二）修改单词：不等号#修改为<>

（1）将字母表中的#注释

（2）在 GetSym()中将<>定义为不等号#

具体操作如下：

```
（1）
SSYM['#']=NEQ;//删去
（2）
else /* THE FOLLOWING TWO CHECK WERE ADDED
             BECAUSE ASCII DOES NOT HAVE A SINGLE CHARACTER FOR <= OR >= */
        if (CH=='<') {
          GetCh();
```

```
        if (CH=='=') { SYM=LEQ; GetCh(); }
        //新增不等号运算符
        else if (CH=='>') { SYM=NEQ;GetCh();}
        else SYM=LSS;
      }
```

（三）增加条件语句的 ELSE 子句，要求写出相关文法，语法描述图，语义描述图

（1）相关文法规则：G(S): S→if S else S | if S | a

（2）语法描述图如图 8.1 所示：

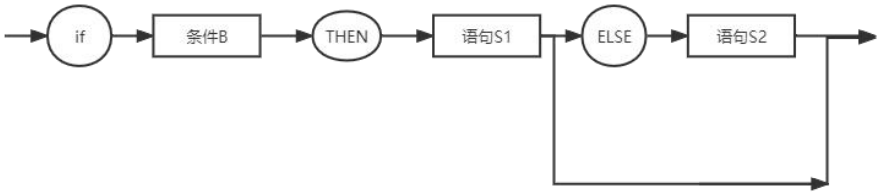


图 8.1 语法描述图

（3）语义描述

```
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();
          CX2=CX;
          GEN(JMP,0,0);        //直接跳转，执行完 Then 后面的则跳转到条件语句最后面
          CODE[CX1].A=CX;      //回填条件跳转，填回 else 语句块中第一句
          STATEMENT(FSYS,LEV,TX);
          CODE[CX2].A=CX;      //回填直接跳转地址
      }
      break;
```

（四）扩充赋值运算*=、/=

（1）在 Statement()函数中的 IDENT 中添加*=和/=的操作

具体操作如下：

```
case IDENT:
      i=POSITION(ID,TX);
      if (i==0) Error(11);
```

```
        else   if   (TABLE[i].KIND!=VARIABLE)   {   /*ASSIGNMENT   TO
NON-VARIABLE*/
            Error(12); i=0;
        }
         GetSym();
        if (SYM==BECOMES) {
            GetSym();
            EXPRESSION(FSYS,LEV,TX);
            if (i!=0) GEN(STO,LEV-TABLE[i].vp.LEVEL,TABLE[i].vp.ADR);
        }
        else if(SYM==TIMESBECOMES) {    // *=
            GEN(LOD,LEV-TABLE[i].vp.LEVEL,TABLE[i].vp.ADR);
            GetSym();
            EXPRESSION(FSYS,LEV,TX);
            GEN(OPR,0,4);
            if (i!=0) GEN(STO,LEV-TABLE[i].vp.LEVEL,TABLE[i].vp.ADR);
        }
        else if(SYM==SLASHBECOMES) {    // /=
            GEN(LOD,LEV-TABLE[i].vp.LEVEL,TABLE[i].vp.ADR);
            GetSym();
            EXPRESSION(FSYS,LEV,TX);
            GEN(OPR,0,5);
            if (i!=0) GEN(STO,LEV-TABLE[i].vp.LEVEL,TABLE[i].vp.ADR);
        }
        else Error(13);
        break;
```

（五）扩充 FOR 语句

（1）在 Statement()函数中的 FORSYM 中添加内容

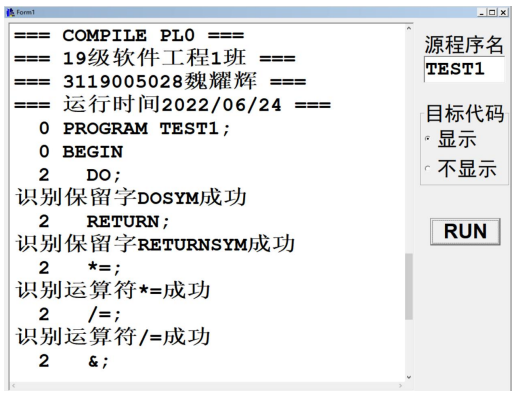
具体操作如下：

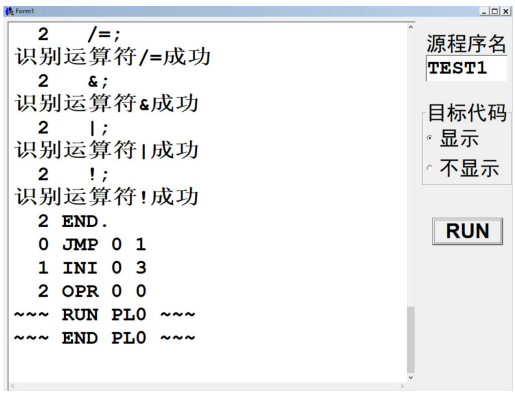
```
case FORSYM:
        GetSym();
        Form1->printfs("识别保留字 FORSYM 成功");
        //GetSym();
        if (SYM != IDENT)
            Error(47);
        else
            i = POSITION(ID, TX);
        if (i == 0)
            Error(11);
        else if (TABLE[i].KIND != VARIABLE)
        {
            Error(12);
            i = 0;
```

```
        }
        GetSym();
        if (SYM == BECOMES)
            GetSym();
        else
            Error(13);
        EXPRESSION(SymSetUnion(SymSetNew(STEPSYM), FSYS), LEV, TX);
        if (i != 0)
            GEN(STO, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        if (SYM == STEPSYM)
            GetSym();
        else
            Error(46);
        CX1 = CX;
        GEN(JMP, 0, 0);
        CX3 = CX;
        EXPRESSION(SymSetUnion(SymSetNew(UNTILSYM), FSYS), LEV, TX);
        GEN(LOD, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        GEN(OPR, 0, 2);
        GEN(STO, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        if (SYM == UNTILSYM)
            GetSym();
        else
            Error(47);
        CODE[CX1].A = CX;
        EXPRESSION(SymSetUnion(SymSetNew(DOSYM), FSYS), LEV, TX);
        GEN(LOD, LEV - TABLE[i].vp.LEVEL, TABLE[i].vp.ADR);
        GEN(OPR, 0, 11);
        CX2 = CX;
        GEN(JPC, 0, 0);
        if (SYM == DOSYM)
            GetSym();
        else
            Error(48);
        STATEMENT(FSYS,LEV,TX);
        GEN(JMP, 0, CX3);
        CODE[CX2].A = CX;
        break;
```

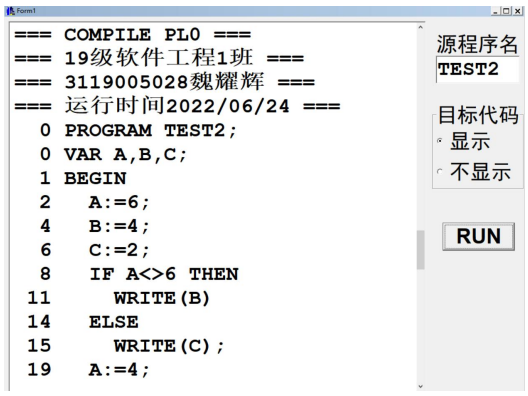
（六）完成结果

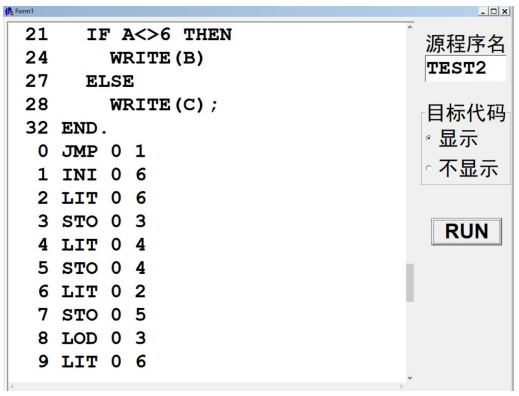
（1）测试用例 1 结果

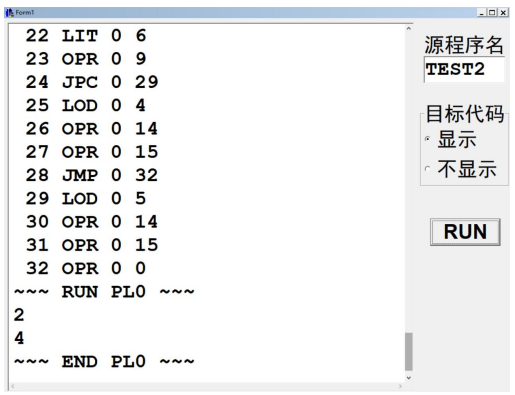




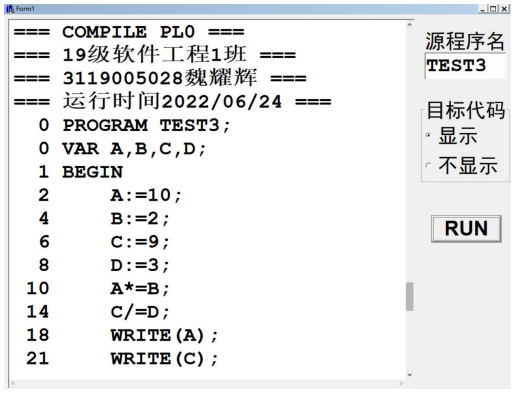
（2）测试用例 2 结果

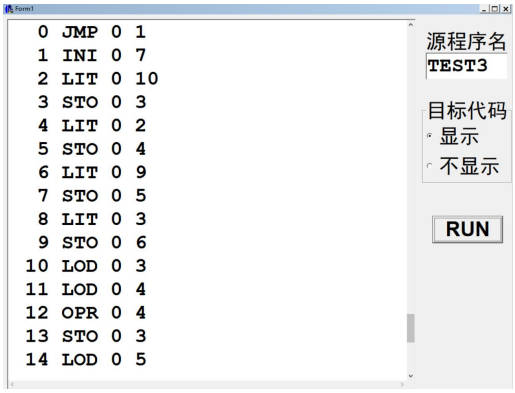


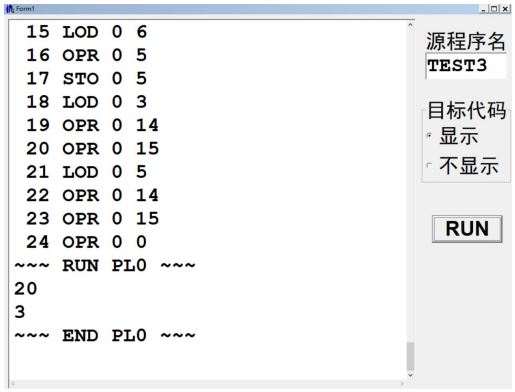




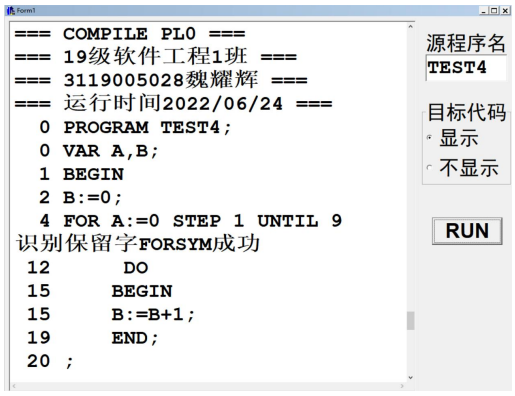
（3）测试用例 3 结果

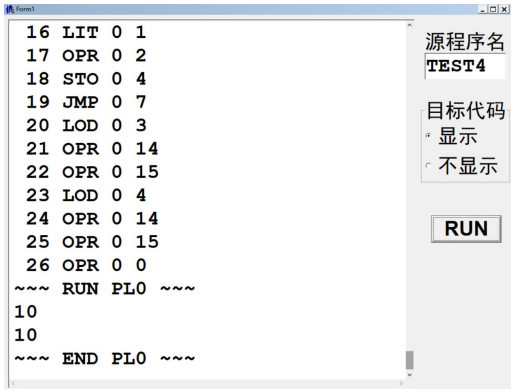






（4）测试用例 4 结果





分析结果可知：

（1）成功识别保留字 ELSE,FOR,STEP,UNTIL,DO,RETURN 以及运算符*=，/=，&，|，！

（2）成功将不等号#改为<>，增加了条件语句的 ELSE 子句

（3）成功扩充*=、/=赋值运算

（4）成功实现 FOR 语句的扩充

九、心得体会

课内的实验使我受益匪浅，通过对 PL/0 编译程序的扩充，我对于语法分析和语义分析有了更加深刻的理解与体会，在完成选做内容的时候，难点在于语义分析的理解以及如何进行语义衔接的跳转，要涉及到整个 PL/0 编译程序的结构以及部分指令的含义才能完成，虽然耗费了不少时间，但让我对编译原理这门课程有了个更加深入的了解。