

报告成绩



《编译原理实验》

学 院 计算机学院

专业班级 17级信息安全2班

学 号 3117005288

姓 名 吴泓川

指导教师 杨劲涛

2020 年 7 月 2日

**课内实验要求**

**对PL/0作以下修改扩充：**

**（1）增加单词：保留字：FOR，TO，DOWNTO，+=，-=，++，--；RETURN**

**要求：词法识别即可**

**（2）替换单词：不等号# 改为 < >，**

**（3）增加条件语句的ELSE子句，要求：写出相关文法，语法图，语义规则。**

1. **环境概述**

实现语言：C/C++

实现工具：Visual Studio 2019

运行平台：windows10

1. **过程或函数的功能表**

|  |  |
| --- | --- |
| 过程或函数名 | 简要功能说明 |
| 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，并送入目标程序区 |
| test | 测试当前单词符号是否合法 |
| block | 分程序分析处理过程 |
| enter | 登录名字表 |
| position(函数) | 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 |

1. **主要成分描述**
2. **符号表**

在编译程序中符号表符号表用来体现作用域与可见性信息，作用是收集符号属性（词法分析）；上下文语义的合法性检查的依据（语法分析）；作为目标代码生成阶段地址分配的依据（语义分析）。符号表中语言符号可分为关键字（保留字）符号，操作符符号及标识符符号。符号表中的标识符一般设置的属性项目有：符号名、符号的类型、符号的存储类别、符号的作用域及可视性符号变量的存储分配信息和符号的其它属性。符号表中所登记的信息在编译的不同阶段都要用到。在语义分析中，符号表所登记的内容将用于语义检查（如检查一个名字的使用和原先的说明是否一致）和产生中间代码。在目标代码生成阶段，当对符号名进行地址分配时，符号表是地址分配的依据。

1. **运行时存储组织和管理**

每个过程的AR有局部变量、中间结果和3个联系单元：

SL： 静态链(存取链)，指向定义该过程的直接外过程（或主程序）运行时最新活动记录的基地址。

DL： 动态链(控制链)，指向调用该过程前正在运行过程的数据段基地址。

RA： 返回地址，记录调用该过程时目标程序的断点，即调用过程指令的下一条指令的地址。

1. **语法分析方法**

PL/0编译程序的语法分析采用了自顶向下的递归的子程序法。语法分析同时也根据程序的语义生成相应三元代码，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）、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）、登录名字表过程（Enter）、查询名字表函数（Position）以及列出类 PCODE代码过程（Listcode）作过语法分析的辅助过程。

程序

分程序

语句

条件

表达式

项

因子

PL/0 语法调用关系图

由PL/0的语法图可知：一个完整的PL/0程序是由分程序和句号构成的。因此，本编译程序在运行的时候，通过主程序中调用分程序处理过程block来分析分程序部分（分程序分析过程中还可能会递归调用block过程），然后，判断最后读入的符号是否为句号。如果是句号且分程序分析中未出错，则是一个合法的PL/0程序，可以运行生成的代码，否则就说明源PL/0程序是不合法的，输出出错提示即可。

1. **中间代码表示**

目标代码类pcode是一种假想栈式计算机的汇编语言。

|  |  |  |
| --- | --- | --- |
| f | L | a |

指令格式：

f为功能码

l为层次差 （标识符引用层减去定义层）

a根据不同的指令有所区别

其功能表为：

|  |  |
| --- | --- |
| lit 0 a | 将常数值取到栈顶，a为常数值 |
| Lod l a | 将变量值取到栈顶，a为偏移量，l为层差 |
| Sto l a | 将栈顶内容送入某变量单元中，a为偏移量，l为层差 |
| Cal a | 调用过程，a为过程地址，l为层差 |
| Int 0 a | 在运行栈中为被调用的过程开辟a个单元的数据区 |
| jmp 0 a | 无条件跳转至a地址 |
| Jpc 0 a | 条件跳转，当栈顶布尔值非真则跳转至a地址，否则顺序执行 |
| opr 0 0 | 过程调用结束后,返回调用点并退栈 |
| opr 0 1 | 栈顶元素取反 |
| opr 0 2 | 次栈顶与栈顶相加，退两个栈元素，结果值进栈 |
| opr 0 3 | 次栈顶减去栈顶，退两个栈元素，结果值进栈 |
| opr 0 4 | 次栈顶乘以栈顶，退两个栈元素，结果值进栈 |
| opr 0 5 | 次栈顶除以栈顶，退两个栈元素，结果值进栈 |
| opr 0 6 | 栈顶元素的奇偶判断，结果值在栈顶 |
| opr 0 7 |  |
| opr 0 8 | 次栈顶与栈顶是否相等，退两个栈元素，结果值进栈 |
| opr 0 9 | 次栈顶与栈顶是否不等，退两个栈元素，结果值进栈 |
| opr 0 10 | 次栈顶是否小于栈顶，退两个栈元素，结果值进栈 |
| opr 0 11 | 次栈顶是否大于等于栈顶，退两个栈元素，结果值进栈 |
| opr 0 12 | 次栈顶是否大于栈顶，退两个栈元素，结果值进栈 |
| opr 0 13 | 次栈顶是否小于等于栈顶，退两个栈元素，结果值进栈 |
| opr 0 14 | 栈顶值输出至屏幕 |
| opr 0 15 | 屏幕输出换行 |
| opr 0 16 | 从命令行读入一个输入置于栈顶 |

1. **实验操作**
2. **增加单词：保留字：FOR，TO，DOWNTO，+=，-=，++，--；RETURN**

在 pl0.h中修改如下：

#define norw 18 /\* 关键字个数 \*/

/\* 符号 \*/

enum symbol {

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,

/\*增加保留字：FOR，TO，DOWNTO，+=，-=，++，--；RETURN\*/

forsym, tosym, downtosym, returnsym, elsesym,

pluseql, mineql, plusplus, minmin

};

#define symnum 41 /\*单词个数\*/

然后在pl0.cpp中的init()函数中修改保留字名字与保留字符号，并按字母顺序排序

/\* 设置保留字名字,按照字母顺序，便于折半查找 \*/

strcpy(&(word[0][0]), "begin");

strcpy(&(word[1][0]), "call");

strcpy(&(word[2][0]), "const");

strcpy(&(word[3][0]), "do");

strcpy(&(word[4][0]), "downto"); /\*增加保留字DOWNTO\*/

strcpy(&(word[5][0]), "else"); /\*增加保留字ELSE\*/

strcpy(&(word[6][0]), "end");

strcpy(&(word[7][0]), "for"); /\*增加保留字FOR\*/

strcpy(&(word[8][0]), "if");

strcpy(&(word[9][0]), "odd");

strcpy(&(word[10][0]), "procedure");

strcpy(&(word[11][0]), "read");

strcpy(&(word[12][0]), "return"); /\*增加保留字RETURN\*/

strcpy(&(word[13][0]), "then");

strcpy(&(word[14][0]), "to"); /\*增加保留字TO\*/

strcpy(&(word[15][0]), "var");

strcpy(&(word[16][0]), "while");

strcpy(&(word[17][0]), "write");

/\* 设置保留字符号 \*/

wsym[0] = beginsym;

wsym[1] = callsym;

wsym[2] = constsym;

wsym[3] = dosym;

wsym[4] = downtosym; /\*增加保留字符号downtosym\*/

wsym[5] = elsesym; /\*增加保留字符号elsesym\*/

wsym[6] = endsym;

wsym[7] = forsym; /\*增加保留字符号forsym\*/

wsym[8] = ifsym;

wsym[9] = oddsym;

wsym[10] = procsym;

wsym[11] = readsym;

wsym[12] = returnsym; /\*增加保留字符号returnsym\*/

wsym[13] = thensym;

wsym[14] = tosym; /\*增加保留字符号tosym\*/

wsym[15] = varsym;

wsym[16] = whilesym;

wsym[17] = writesym;

之后便是词法分析部分

添加++、--、+=、-=识别，需要在getsym()中添加

//添加++，+=，--，-=

else if (ch == '+')

{

getchdo;

if (ch == '=') //增加+=

{

sym = pluseql;

getchdo;

}

else if (ch == '+') //增加++

{

sym = plusplus;

getchdo;

}

else sym = plus;

}

else if (ch == '-')

{

getchdo;

if (ch == '=') //增加-=

{

sym = mineql;

getchdo;

}

else if (ch == '-') //增加--

{

sym = minmin;

getchdo;

}

else sym = minus;

}

成功添加FOR，TO，DOWNTO，+=，-=，++，--；RETURN。

1. **替换单词：不等号# 改为 < >**

在init()函数中将ssym['#'] = neq;注释掉

然后在GetSym()修改为

if (ch == '<') /\* 检测小于或小于等于符号 \*/

{

getchdo;

if (ch == '=')

{

sym = leq;

getchdo;

}

else

{

if (ch == '>') //增加<>为不等号

{

sym = neq;

getchdo;

}

else sym = lss;

}

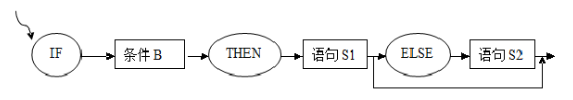
}

则成功将#变为<>

1. **增加条件语句的ELSE子句，要求：写出相关文法，语法图，语义规则。**
2. **相关文法**

G(S): S→if S else S | if S | a

1. **语法图**



1. **语义规则**

|  |  |
| --- | --- |
| 产生式 | 语义规则 |
| S→if B then M S1 | { backpatch(E.truelist, M.quad );  S.nextlist:=merge(E.falselist, S1.nextlist) } |
| M → ε | { M.quad := nextquad ; } |
| N → ε | { N.nextlist:=makelist(nextquad);  Gen( j , — , — , 0 ) } |
| S→if B then M1 S1  N else M2 S2 | { backpatch(E.truelist, M1.quad );  backpatch(E.falselist, M2.quad );  S.nextlist:=merge(S1.nextlist,  N.nextlist, S2.nextlist) } |

**代码修改：**

**修改前：**

if (sym == ifsym) /\* 准备按照if语句处理 \*/

{

getsymdo;

memcpy(nxtlev, fsys, sizeof(bool) \* symnum);

nxtlev[thensym] = true;

nxtlev[dosym] = true; /\* 后跟符号为then或do \*/

conditiondo(nxtlev, ptx, lev); /\* 调用条件处理（逻辑运算）函数 \*/

if (sym == thensym)

{

getsymdo;

}

else

{

error(16); /\* 缺少then \*/

cx1 = cx; /\* 保存当前指令地址 \*/

gendo(jpc, 0, 0); /\* 生成条件跳转指令，跳转地址未知，暂时写0 \*/

statementdo(fsys, ptx, lev); /\* 处理then后的语句 \*/

break;

**修改后：**

if (sym == ifsym) /\* 准备按照if语句处理 \*/

{

getsymdo;

memcpy(nxtlev, fsys, sizeof(bool) \* symnum);

nxtlev[thensym] = true;

nxtlev[dosym] = true; /\* 后跟符号为then或do \*/

conditiondo(nxtlev, ptx, lev); /\* 调用条件处理（逻辑运算）函数 \*/

if (sym == thensym)

{

getsymdo;

}

else

{

error(16); /\* 缺少then \*/

}

cx1 = cx; /\* 保存当前指令地址 \*/

gendo(jpc, 0, 0); /\* 生成条件跳转指令，跳转地址未知，暂时写0 \*/

statementdo(fsys, ptx, lev); /\* 处理then后的语句 \*/

if(sym!=elsesym) code[cx1].a = cx; /\* 经statement处理后，cx为then后语句执行完的位置，它正是前面未定的跳转地址 \*/

//如果程序中没有else语句，执行JPC时跳到此地址

else {

//添加ELSE语句

getsymdo;

cx2 = cx;

gendo(jmp, 0, 0); //跳过else语句

code[cx1].a = cx;

//如果程序中有else语句，执行JPC时跳到此地址

statementdo(fsys, ptx, lev); //执行语句

code[cx2].a = cx; //结束指令的地址回填到无条件跳转的地址

}

}

1. **测试用例**

**（1）用例1**

var a;

begin

for a:=1 to 8 do

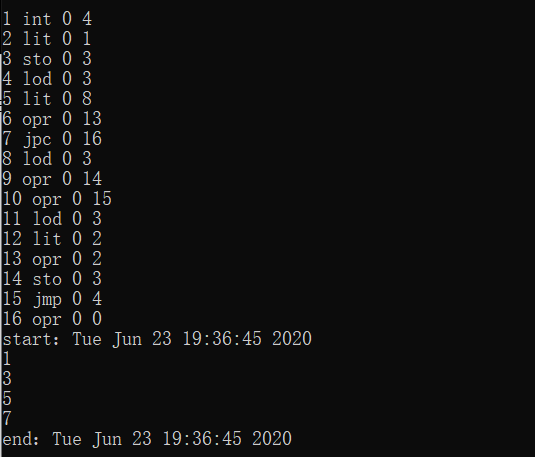
write(a);

end.

**实验结果**

识别for，to





**（2）用例2**

var a;

begin

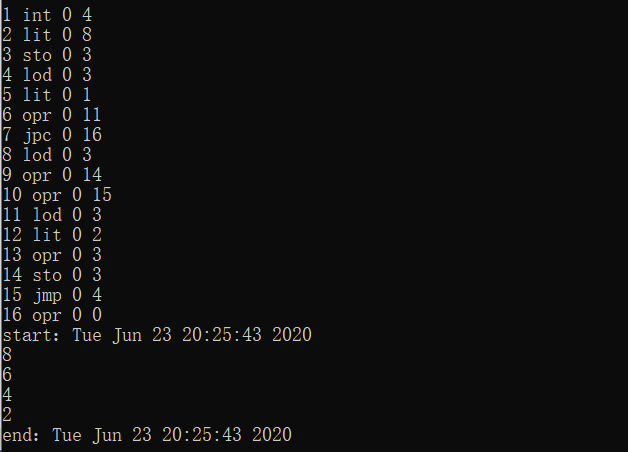
for a:=8 downto 1 do

write(a);

end.

识别for，downto





**（3）用例3**

var a,b;

begin

a:=3;

b:=2;

a++;

b--;

a+=1;

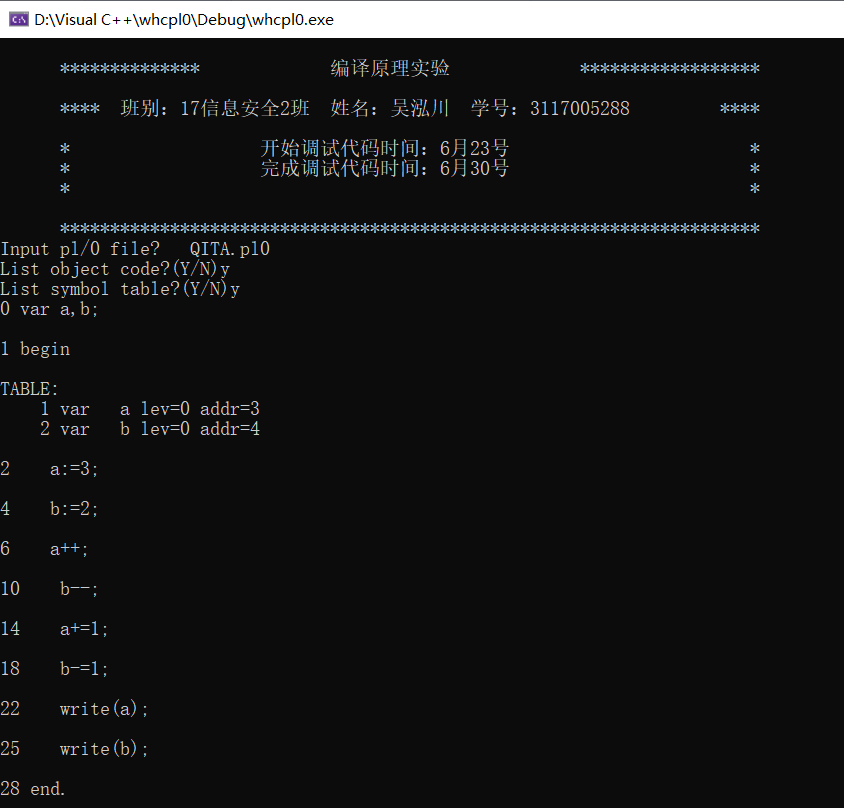
b-=1;

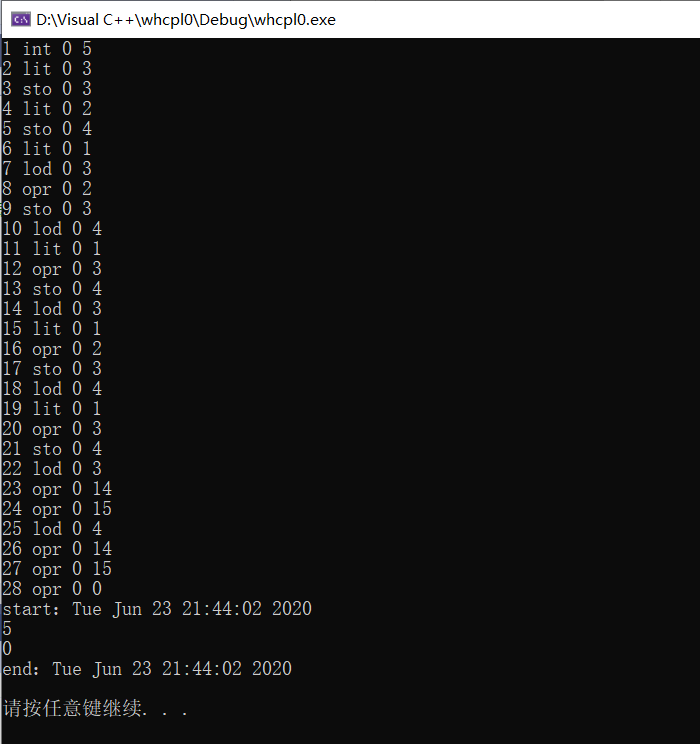
write(a);

write(b);

end.

识别++、--、+=、-=





**（4）用例4**

var a;

begin

a:=1;

if a<>1 then

a:=2;

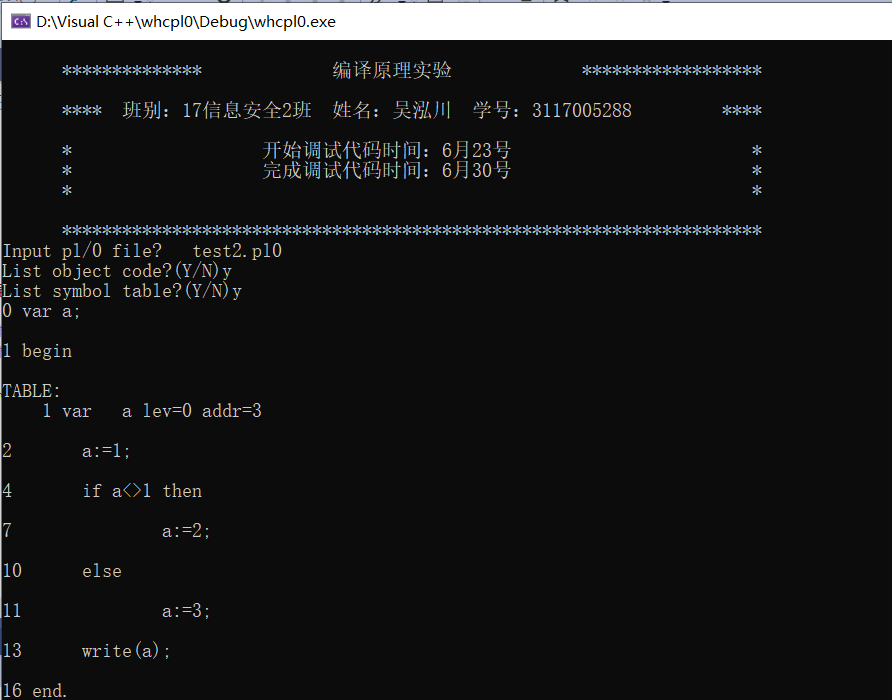
else

a:=3;

write(a);

end.

**实验结果**





由此可见，成功将<>识别为不等号，并运行else子句，并输出真正的a值3。