# 实验三 PLO编译器的了解

### 一、实验目的

1 掌握PLO语言的文法

2 PLO语言编译器程序条件语句的结构

3 掌握PLO语言编译器对循环语句的编译过程

4 掌握计数型循环语句的编译过程

### 二、实验预习

1、PLO语言编译器程序

2、Pl0语言条件语句文法定义

3、循环语句的编译过程

4、计数型循环语句的文法定义

### 三、实验内容和要求

1，用DEV C++ 打开PL0.c,了解源程序的总体结构及各模块功能、调用关系，掌握IF语句的程序模块

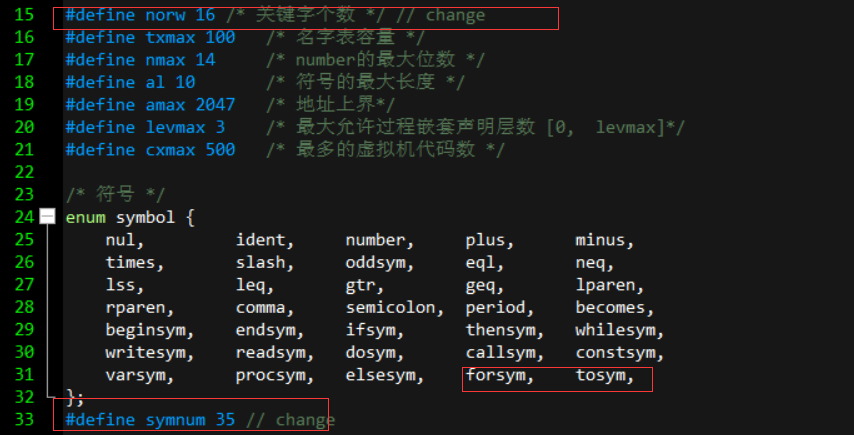
2、修改pl0.c编译程序，扩充语言计数型循环语句成份for

3、计数型循环语句的方法定义

如for 变量=初值 to 终值 语句系列

四、实验过程

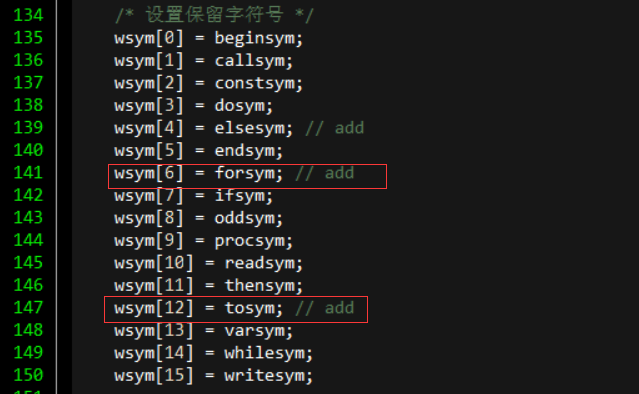
修改pl0.h中符号和保留字个数



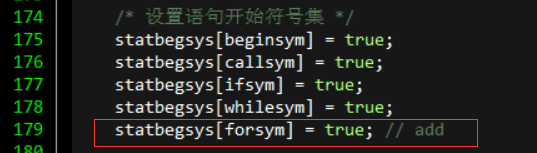
添加符号



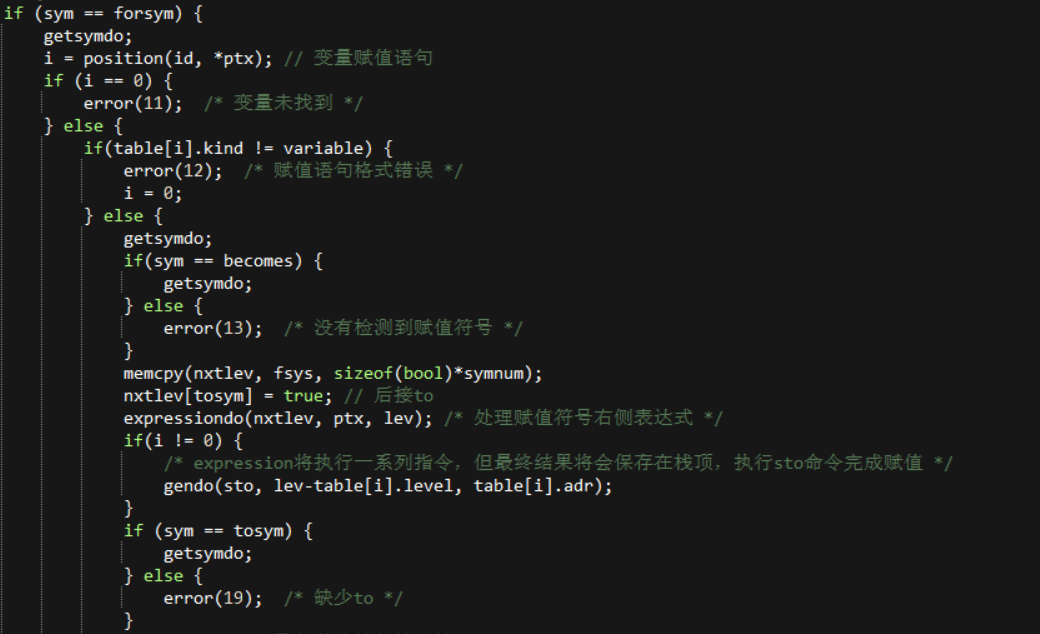
添加保留字



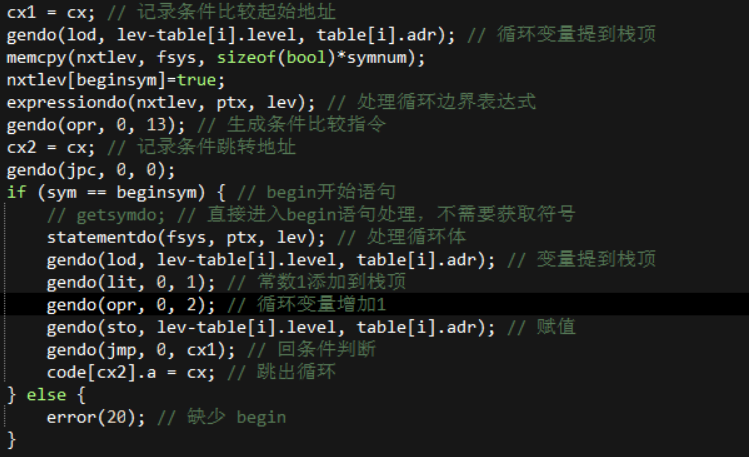
设置语句开始符号集



处理 for i:=1 to



处理后续内容和循环体



完整代码：

if (sym == forsym) {

getsymdo;

i = position(id, \*ptx); // 变量赋值语句

if (i == 0) {

error(11); /\* 变量未找到 \*/

} else {

if(table[i].kind != variable) {

error(12); /\* 赋值语句格式错误 \*/

i = 0;

} else {

getsymdo;

if(sym == becomes) {

getsymdo;

} else {

error(13); /\* 没有检测到赋值符号 \*/

}

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

nxtlev[tosym] = true; // 后接to

expressiondo(nxtlev, ptx, lev); /\* 处理赋值符号右侧表达式 \*/

if(i != 0) {

/\* expression将执行一系列指令，但最终结果将会保存在栈顶，执行sto命令完成赋值 \*/

gendo(sto, lev-table[i].level, table[i].adr);

}

if (sym == tosym) {

getsymdo;

} else {

error(19); /\* 缺少to \*/

}

cx1 = cx; // 记录条件比较起始地址

gendo(lod, lev-table[i].level, table[i].adr); // 循环变量提到栈顶

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

nxtlev[beginsym]=true;

expressiondo(nxtlev, ptx, lev); // 处理循环边界表达式

gendo(opr, 0, 13); // 生成条件比较指令

cx2 = cx; // 记录条件跳转地址

gendo(jpc, 0, 0);

if (sym == beginsym) { // begin开始语句

// getsymdo; // 直接进入begin语句处理，不需要获取符号

statementdo(fsys, ptx, lev); // 处理循环体

gendo(lod, lev-table[i].level, table[i].adr); // 变量提到栈顶

gendo(lit, 0, 1); // 常数1添加到栈顶

gendo(opr, 0, 2); // 循环变量增加1

gendo(sto, lev-table[i].level, table[i].adr); // 赋值

gendo(jmp, 0, cx1); // 回条件判断

code[cx2].a = cx; // 跳出循环

} else {

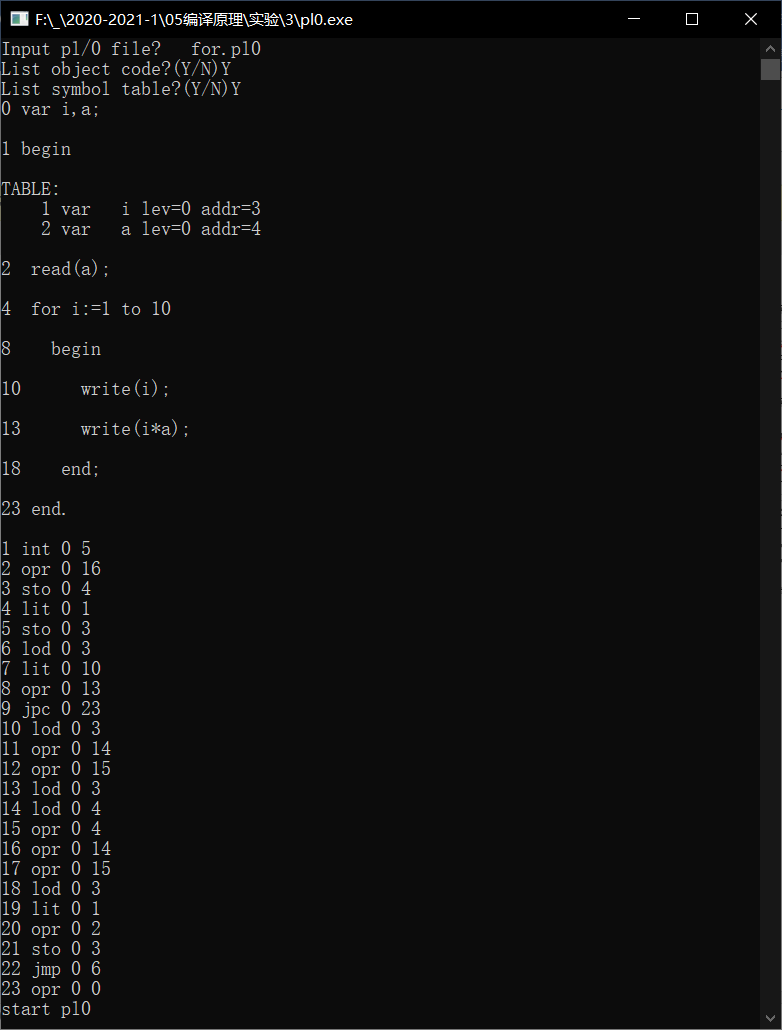
error(20); // 缺少 begin

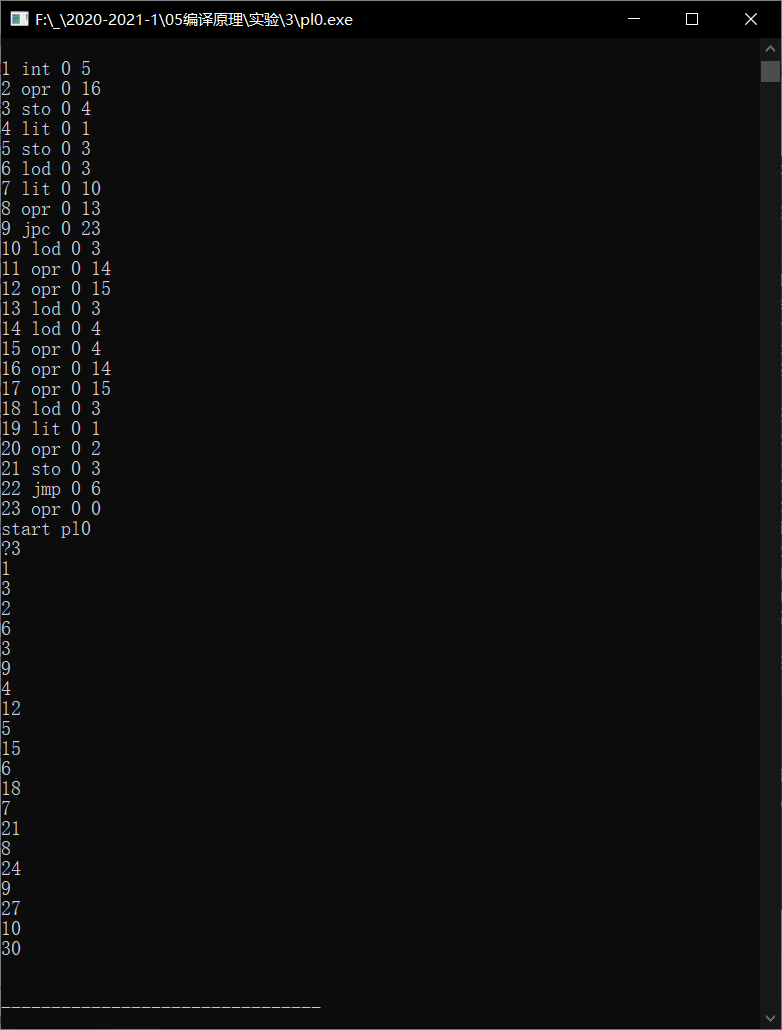
}

}

}

运行结果





**思考题：**