{\* 注释者：周美廷-76066002 \*}

{\* 程序明为pl0 \*}

program pl0(input,output,fin) ; { version 1.0 oct.1989 }

{ PL/0 compiler with code generation }

{\* 常量定义 \*}

{\* 保留字数 \*}

const norw = 13; { no. of reserved words }

{\* 标识符表的长度 \*}

txmax = 100; { length of identifier table }

{\* 最大数字位数 \*}

nmax = 14; { max. no. of digits in numbers }

{\* 标识符的长度 \*}

al = 10; { length of identifiers }

{\* 最大地址 \*}

amax = 2047; { maximum address }

{\* 块嵌套的最大深度 \*}

levmax = 3; { maximum depth of block nesting }

{\* 代码数组的大小 \*}

cxmax = 200; { size of code array }

{\* 类型变量定义 \*}

type 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,dosym,callsym,constsym,

varsym,procsym,readsym,writesym );

alfa = packed array[1..al] of char;

objecttyp = (constant,variable,prosedure);

symset = set of symbol;

fct = ( lit,opr,lod,sto,cal,int,jmp,jpc,red,wrt ); { functions }

instruction = packed record

f : fct; { function code }

l : 0..levmax; { level }

a : 0..amax; { displacement address }

end;

{ lit 0, a : load constant a

opr 0, a : execute operation a

lod l, a : load variable l,a

sto l, a : store variable l,a

cal l, a : call procedure a at level l

int 0, a : increment t-register by a

jmp 0, a : jump to a

jpc 0, a : jump conditional to a

red l, a : read variable l,a

wrt 0, 0 : write stack-top

}

{\* 变量定义 \*}

var ch : char; { last character read }

sym: symbol; { last symbol read }

id : alfa; { last identifier read }

num: integer; { last number read }

cc : integer; { character count }

ll : integer; { line length }

kk,err: integer;

cx : integer; { code allocation index }

line: array[1..81] of char;

a : alfa;

code : array[0..cxmax] of instruction;

word : array[1..norw] of alfa;

wsym : array[1..norw] of symbol;

ssym : array[char] of symbol;

mnemonic : array[fct] of

packed array[1..5] of char;

declbegsys, statbegsys, facbegsys : symset;

table : array[0..txmax] of

record

name : alfa;

case kind: objecttyp of

constant : (val:integer );

variable,prosedure: (level,adr: integer )

end;

fin : text; { source program file }

sfile: string; { source program file name }

{\* 过程：error（参数n）（处理错误） \*}

procedure error( n : integer );

begin

writeln( '\*\*\*\*', ' ':cc-1, '^', n:2 );

{\* 错误次数加1 \*}

err := err+1

end; { error }

{\* 过程：getsym （词法分析）\*}

procedure getsym;

var i,j,k : integer;

{\* 过程：getch \*}

procedure getch;

begin

{\* 若计数变量已到行结尾 \*}

if cc = ll { get character to end of line }

then begin { read next line }

{\* 如果已达到fin的文件结尾，那么就是错误（错误处理） \*}

if eof(fin)

then begin

writeln('program incomplete');

{\* 把fin文件关掉 \*}

close(fin);

exit;

end;

{\* 清零（重启cc和ll） \*}

ll := 0;

cc := 0;

write(cx:4,' '); { print code address }

{\* while 还没到fin行的结束 \*}

while not eoln(fin) do

begin

{\* 当前行长度加1 \*}

ll := ll+1;

{\* 读取fin中的字符放在ch \*}

read(fin,ch);

{\* 输出ch \*}

write(ch);

{\* 当前行的字符值为ch的值\*}

line[ll] := ch

end;

{\* 输出新行 \*}

writeln;

{\* 读取psin的当前一行，鼠标放在下一行 \*}

readln(fin);

ll := ll+1;

line[ll] := ' ' { process end-line }

end;

{\* 计数变量加1 \*}

cc := cc+1;

{\* ch赋值line[cc]的值 \*}

ch := line[cc]

end; { getch }

begin { procedure getsym; }

{\* while ch 是空格，读下一个字符 \*}

while ch = ' ' do

getch;

{\* 若ch是字母 \*}

if ch in ['a'..'z']

then begin { identifier of reserved word }

k := 0;

{\* 重复以下程序直到ch不是字母也不是数字 \*}

repeat

{\* 若果k还不等于标识符的长度，k+1，把ch放在a数组里第k对应的位置 \*}

if k < al

then begin

k := k+1;

a[k] := ch

end;

{\* 读下一个字符 \*}

getch

until not( ch in ['a'..'z','0'..'9']);

{\* 若k等于或超过kk，那把k的值赋给kk \*}

if k >= kk { kk : last identifier length }

then kk := k

{\* 不然重复以下程序直到kk等于k \*}

else repeat

{\* a数组第kk元素为空格，然后kk-1 \*}

a[kk] := ' ';

kk := kk-1

until kk = k;

id := a;

i := 1;

j := norw; { binary search reserved word table }

{\* 重复以下程序直到i大于j \*}

repeat

k := (i+j) div 2;

if id <= word[k]

then j := k-1;

if id >= word[k]

then i := k+1

until i > j;

{\* 若i-1>j，sym是wsym[k]对应的符号 \*}

if i-1 > j

then sym := wsym[k]

{\* 不然sym是标识符 \*}

else sym := ident

end

{\* 若ch是数字 \*}

else if ch in ['0'..'9']

then begin { number }

{\* 初始化k和num，sym是数字 \*}

k := 0;

num := 0;

sym := number;

{\* 重复以下程序直到ch不是数字 \*}

repeat

num := 10\*num+(ord(ch)-ord('0'));

k := k+1;

{\* 读下一个字符 \*}

getch

until not( ch in ['0'..'9']);

{\* 若k>nmax那就是第30错误 \*}

if k > nmax

then error(30)

end

{\* 若ch是colon，而下一个字符是等号，sym是becomes然后读下一个字符，不然sym是null\*}

else if ch = ':'

then begin

getch;

if ch = '='

then begin

sym := becomes;

getch

end

else sym := nul

end

{\* 如果ch是小于号，然后下一个ch是等号，那么sym是leq然后继续读下一个字符，

而如果下一个是大于号，那么sym是neq然后继续读下一个字符，不然sy是lss \*}

else if ch = '<'

then begin

getch;

if ch = '='

then begin

sym := leq;

getch

end

else if ch = '>'

then begin

sym := neq;

getch

end

else sym := lss

end

{\* 如果ch是大于号，然后下一个ch是等号，那么sym是geq然后继续读下一个字符，

不然sy是gtr \*}

else if ch = '>'

then begin

getch;

if ch = '='

then begin

sym := geq;

getch

end

else sym := gtr

end

{\* 不然sym就是ssym[ch]对应的符号，继续读下一个字符 \*}

else begin

sym := ssym[ch];

getch

end

end; { getsym }

{\* 过程：gen（参数x，y，z） \*}

procedure gen( x: fct; y,z : integer );

begin

if cx > cxmax

then begin

writeln('program too long');

close(fin);

exit

end;

with code[cx] do

begin

f := x;

l := y;

a := z

end;

cx := cx+1

end; { gen }

{\* 过程：test（参数s1，s2，n）（试试看sym是否在s1里，不然就是不合法，跳过） \*}

procedure test( s1,s2 :symset; n: integer );

begin

{\* 如果sy不在s1里，那就要第n错误 \*}

if not ( sym in s1 )

then begin

error(n);

s1 := s1+s2;

{\* while sym不在s1中，继续读下一个符号 \*}

while not( sym in s1) do

getsym

end

end; { test }

{\* 过程：block（参数lev，tx，fsys） \*}

procedure block( lev,tx : integer; fsys : symset );

var dx : integer; { data allocation index }

tx0: integer; { initial table index }

cx0: integer; { initial code index }

{\* 过程：enter（参数k） \*}

procedure enter( k : objecttyp );

begin { enter object into table }

tx := tx+1;

{\* 对于表中第tx对应的元素做以下操作 \*}

with table[tx] do

begin

name := id;

kind := k;

case k of

{\* 若k是常量，若num大于amax，那就是第30错误，num为0 \*}

constant : begin

if num > amax

then begin

error(30);

num := 0

end;

val := num

end;

{\* 若k是变量 \*}

variable : begin

level := lev;

adr := dx;

dx := dx+1

end;

{\* 若k是过程 \*}

prosedure: level := lev;

end

end

end; { enter }

{\* 函数：position（参数id） \*}

function position ( id : alfa ): integer;

var i : integer;

begin

{\* 表中第0元素的名字赋值id的值 \*}

table[0].name := id;

i := tx;

{\* while表中第i元素的名字不是id，i-1 \*}

while table[i].name <> id do

i := i-1;

{\* 更新position为i \*}

position := i

end; { position }

{\* 过程：constdeclaration \*}

procedure constdeclaration;

begin

{\* 若sym是标识符，读下一个符号 \*}

if sym = ident

then begin

getsym;

{\* 若sym属于eql，becomes \*}

if sym in [eql,becomes]

then begin

{\* 若sym是becomes，那就是第1错误 \*}

if sym = becomes

then error(1);

{\* 读下一个符号，若sym是数字，把常量放入表中然后读下一个符号 \*}

getsym;

if sym = number

then begin

enter(constant);

getsym

end

{\* 不然就是第2错误 \*}

else error(2)

end

{\* 不然就是第3错误 \*}

else error(3)

end

{\* 不然就是第4错误 \*}

else error(4)

end; { constdeclaration }

{\* 过程：vardeclaration \*}

procedure vardeclaration;

begin

{\* 若sym是标识符 \*}

if sym = ident

then begin

{\* 把变量放入表中然后读下一个符号 \*}

enter(variable);

getsym

end

{\* 不然就是第4错误 \*}

else error(4)

end; { vardeclaration }

{\* 过程：listcode \*}

procedure listcode;

var i : integer;

begin

for i := cx0 to cx-1 do

with code[i] do

writeln( i:4, mnemonic[f]:7,l:3, a:5)

end; { listcode }

{\* 过程：statement（参数fsys） \*}

procedure statement( fsys : symset );

var i,cx1,cx2: integer;

{\* 过程：expression（参数fsys） \*}

procedure expression( fsys: symset);

var addop : symbol;

{\* 过程：term（参数fsys） \*}

procedure term( fsys : symset);

var mulop: symbol ;

{\* 过程：factor（参数fsys） \*}

procedure factor( fsys : symset );

var i : integer;

begin

test( facbegsys, fsys, 24 );

while sym in facbegsys do

begin

{\* 若sym是标识符 \*}

if sym = ident

then begin

{\* 在表中寻找id的位置放在i中 \*}

i := position(id);

{\* 若i是0那就是没找到，那就是第11错误 \*}

if i= 0

then error(11)

else

{\* 对于表中第i元素做以下操作 \*}

with table[i] do

case kind of

{\* 若类型是constant \*}

constant : gen(lit,0,val);

{\* 若类型是变量 \*}

variable : gen(lod,lev-level,adr);

{\* 若类型是过程 \*}

prosedure: error(21)

end;

{\* 读下一个符号 \*}

getsym

end

{\* 若sym是数字 \*}

else if sym = number

then begin

{\* 若num大于amax，那就是第30错误，num为0 \*}

if num > amax

then begin

error(30);

num := 0

end;

gen(lit,0,num);

{\* 读下一个符号 \*}

getsym

end

{\* 若sym是左括号，读下一个符号 \*}

else if sym = lparen

then begin

getsym;

expression([rparen]+fsys);

{\* 若sym是右括号，读下一个符号，不然就是第22错误 \*}

if sym = rparen

then getsym

else error(22)

end;

test(fsys,[lparen],23)

end

end; { factor }

begin { procedure term( fsys : symset);

var mulop: symbol ; }

factor( fsys+[times,slash]);

{\* while sym属于times，slash \*}

while sym in [times,slash] do

begin

mulop := sym;

{\* 读下一个符号 \*}

getsym;

factor( fsys+[times,slash] );

{\* 若mulop是times \*}

if mulop = times

then gen( opr,0,4 )

else gen( opr,0,5)

end

end; { term }

begin { procedure expression( fsys: symset);

var addop : symbol; }

{\* 若sym属于plus，minus \*}

if sym in [plus, minus]

then begin

addop := sym;

{\* 读下一个符号 \*}

getsym;

term( fsys+[plus,minus]);

if addop = minus

then gen(opr,0,1)

end

else term( fsys+[plus,minus]);

{\* while sym属于plus，minus \*}

while sym in [plus,minus] do

begin

addop := sym;

{\* 读下一个符号 \*}

getsym;

term( fsys+[plus,minus] );

if addop = plus

then gen( opr,0,2)

else gen( opr,0,3)

end

end; { expression }

{\* 过程：condition（参数fsys） \*}

procedure condition( fsys : symset );

var relop : symbol;

begin

{\* 若sym是oddsym，读下一个符号 \*}

if sym = oddsym

then begin

getsym;

expression(fsys);

gen(opr,0,6)

end

else begin

expression( [eql,neq,lss,gtr,leq,geq]+fsys);

{\* 若sym不属于eql,neq,lss,leq,gtr,geq，那就是第20错误 \*}

if not( sym in [eql,neq,lss,leq,gtr,geq])

then error(20)

else begin

{\* relop为sym的值 \*}

relop := sym;

{\* 读下一个符号 \*}

getsym;

expression(fsys);

{\* 对于relop不一样的属性，有不一样的处理 \*}

case relop of

eql : gen(opr,0,8);

neq : gen(opr,0,9);

lss : gen(opr,0,10);

geq : gen(opr,0,11);

gtr : gen(opr,0,12);

leq : gen(opr,0,13);

end

end

end

end; { condition }

begin { procedure statement( fsys : symset );

var i,cx1,cx2: integer; }

{\* 若sym是标识符 \*}

if sym = ident

then begin

{\* 在表中寻找id的位置放在i \*}

i := position(id);

{\* 若i等于0那就是没找到，那就是第11错误 \*}

if i= 0

then error(11)

{\* 若表中第i元素的类型不是变量，那就是第12错误，i为0 \*}

else if table[i].kind <> variable

then begin { giving value to non-variation }

error(12);

i := 0

end;

{\* 读下一个符号 \*}

getsym;

{\* 若sym是becomes，读下一个符号，不然是第13错误 \*}

if sym = becomes

then getsym

else error(13);

expression(fsys);

{\* 若i不为0 \*}

if i <> 0

then

with table[i] do

gen(sto,lev-level,adr)

end

{\* 若sym是callsym，读下一个符号 \*}

else if sym = callsym

then begin

getsym;

{\* 若sym不是标识符，那就是第14错误 \*}

if sym <> ident

then error(14)

else begin

{\* 在表中寻找id的位置放在i \*}

i := position(id);

{\* 若i是0，那就是没找到，那就是第11错误 \*}

if i = 0

then error(11)

else

{\* 对于表中第i元素做以下操作 \*}

with table[i] do

{\* 若类型是过程 \*}

if kind = prosedure

then gen(cal,lev-level,adr)

{\* 不然就是第15错误 \*}

else error(15);

{\* 读下一个符号 \*}

getsym

end

end

{\* 若sym是ifsym，读下一个符号 \*}

else if sym = ifsym

then begin

getsym;

condition([thensym,dosym]+fsys);

{\* 若sym是thensym，读下一个符号，不然就是第16错误 \*}

if sym = thensym

then getsym

else error(16);

cx1 := cx;

gen(jpc,0,0);

statement(fsys);

code[cx1].a := cx

end

{\* 若sym是beginsym，读下一个符号 \*}

else if sym = beginsym

then begin

getsym;

statement([semicolon,endsym]+fsys);

{\* while sym属于[semicolon]+statbegsys \*}

while sym in ([semicolon]+statbegsys) do

begin

{\* 若sym是semicolon，读下一个符号，不然就是第10错误 \*}

if sym = semicolon

then getsym

else error(10);

statement([semicolon,endsym]+fsys)

end;

{\* 若sym是endsym，读下一个符号，不然就是第17错误 \*}

if sym = endsym

then getsym

else error(17)

end

{\* 若sym是whiltesym \*}

else if sym = whilesym

then begin

cx1 := cx;

{\* 读下一个符号 \*}

getsym;

condition([dosym]+fsys);

cx2 := cx;

gen(jpc,0,0);

{\* 若sym是dosym，读下一个符号，不然就是第18错误 \*}

if sym = dosym

then getsym

else error(18);

statement(fsys);

gen(jmp,0,cx1);

code[cx2].a := cx

end

{\* 若sym是readsym，读下一个符号 \*}

else if sym = readsym

then begin

getsym;

{\* 若sym是左括号 \*}

if sym = lparen

then

{\* 重复以下程序直到sym不是逗号 \*}

repeat

getsym;

{\* 若sym是标识符 \*}

if sym = ident

then begin

{\* 在表中寻找id的位置放在i \*}

i := position(id);

{\* 若i是0，那就是没找到，那时第11错误 \*}

if i = 0

then error(11)

{\* 若表中第i元素对应的类型不是变量 \*}

else if table[i].kind <> variable

then begin

{\* 那就是第12错误，i为0 \*}

error(12);

i := 0

end

else with table[i] do

gen(red,lev-level,adr)

end

{\* 不然就是第4错误 \*}

else error(4);

{\* 读下一个符号 \*}

getsym;

until sym <> comma

{\* 不然就是第40错误 \*}

else error(40);

{\* 若sym不是右括号，那就是第22错误 \*}

if sym <> rparen

then error(22);

{\* 读下一个符号 \*}

getsym

end

{\* 若sym是writesym，读下一个符号 \*}

else if sym = writesym

then begin

getsym;

{\* 若sym是左括号 \*}

if sym = lparen

then begin

{\* 重复以下程序直到sym不是逗号 \*}

repeat

{\* 读下一个符号 \*}

getsym;

expression([rparen,comma]+fsys);

gen(wrt,0,0);

until sym <> comma;

{\* 若sym不是右括号，那就是第22错误 \*}

if sym <> rparen

then error(22);

{\* 读下一个符号 \*}

getsym

end

{\* 不然就是第40错误 \*}

else error(40)

end;

test(fsys,[],19)

end; { statement }

begin { procedure block( lev,tx : integer; fsys : symset );

var dx : integer; /\* data allocation index \*/

tx0: integer; /\*initial table index \*/

cx0: integer; /\* initial code index \*/ }

dx := 3;

tx0 := tx;

table[tx].adr := cx;

gen(jmp,0,0); { jump from declaration part to statement part }

if lev > levmax

then error(32);

{\* 重复以下程序直到sym不属于declbegsys \*}

repeat

{\* 若sym是constsym，读下一个符号 \*}

if sym = constsym

then begin

getsym;

{\* 重复以下程序直到sym不是标识符 \*}

repeat

constdeclaration;

{\* 若sym是逗号，读下一个符号 \*}

while sym = comma do

begin

getsym;

constdeclaration

end;

{\* 若sym是semicolon，读下一个符号，那就是第5错误 \*}

if sym = semicolon

then getsym

else error(5)

until sym <> ident

end;

{\* 若sym是varsym，读下一个符号 \*}

if sym = varsym

then begin

getsym;

{\* 重复以下程序直到sym不是标识符 \*}

repeat

vardeclaration;

{\* 若sym是逗号，读下一个符号 \*}

while sym = comma do

begin

getsym;

vardeclaration

end;

{\* 若sym是semicolon，读下一个符号，那就是第5错误 \*}

if sym = semicolon

then getsym

else error(5)

until sym <> ident;

end;

{\* while sym是procsym，读下一个符号 \*}

while sym = procsym do

begin

getsym;

{\* 若sym是标识符，把过程放在表中，然后读下一个符号 \*}

if sym = ident

then begin

enter(prosedure);

getsym

end

{\* 不然就是第4错误 \*}

else error(4);

{\* 若sym是semicolon，读下一个符号，不然就是第5错误 \*}

if sym = semicolon

then getsym

else error(5);

block(lev+1,tx,[semicolon]+fsys);

{\* 若sym是semicolon，读下一个符号 \*}

if sym = semicolon

then begin

getsym;

test( statbegsys+[ident,procsym],fsys,6)

end

{\* 不然就是第5错误 \*}

else error(5)

end;

test( statbegsys+[ident],declbegsys,7)

until not ( sym in declbegsys );

code[table[tx0].adr].a := cx; { back enter statement code's start adr. }

with table[tx0] do

begin

adr := cx; { code's start address }

end;

cx0 := cx;

gen(int,0,dx); { topstack point to operation area }

statement( [semicolon,endsym]+fsys);

gen(opr,0,0); { return }

test( fsys, [],8 );

listcode;

end { block };

{\* 过程：interpret（解释执行） \*}

procedure interpret;

const stacksize = 500; {\* 栈大小定义 \*}

var p,b,t: integer; { program-,base-,topstack-register }

i : instruction;{ instruction register }

s : array[1..stacksize] of integer; { data store }

{\* 函数：base（参数l） \*}

function base( l : integer ): integer;

var b1 : integer;

begin { find base l levels down }

b1 := b;

{\* while层数大于0 \*}

while l > 0 do

begin

{\* 记录当前层数据基址的内容，l-1 \*}

b1 := s[b1];

l := l-1

end;

{\* 把找到的基地址保存 \*}

base := b1

end; { base }

begin

writeln( 'START PL/0' );

t := 0;

b := 1;

p := 0;

s[1] := 0;

s[2] := 0;

s[3] := 0;

{\* 重复以下程序直到p等于0 \*}

repeat

i := code[p];

p := p+1;

with i do

case f of

{\* 若f是lit \*}

lit : begin

t := t+1;

s[t]:= a;

end;

{\* 若f是opr \*}

opr : case a of { operator }

{\* 若是第0操作那就返回 \*}

0 : begin { return }

t := b-1;

p := s[t+3];

b := s[t+2];

end;

{\* 若是第1操作那就取反 \*}

1 : s[t] := -s[t];

{\* 若是第2操作那就求和 \*}

2 : begin

t := t-1;

s[t] := s[t]+s[t+1]

end;

{\* 若是第3操作那就做差 \*}

3 : begin

t := t-1;

s[t] := s[t]-s[t+1]

end;

{\* 若是第4操作那就乘积 \*}

4 : begin

t := t-1;

s[t] := s[t]\*s[t+1]

end;

{\* 若是第5操作那就相处 \*}

5 : begin

t := t-1;

s[t] := s[t]div s[t+1]

end;

{\* 若是第6操作那就判断奇数 \*}

6 : s[t] := ord(odd(s[t]));

{\* 若是第8操作那就判断等值 \*}

8 : begin

t := t-1;

s[t] := ord(s[t]=s[t+1])

end;

{\* 若是第9操作那就判断不等值 \*}

9 : begin

t := t-1;

s[t] := ord(s[t]<>s[t+1])

end;

{\* 若是第9操作那就判断小于值 \*}

10: begin

t := t-1;

s[t] := ord(s[t]< s[t+1])

end;

{\* 若是第9操作那就判断大于等于值 \*}

11: begin

t := t-1;

s[t] := ord(s[t] >= s[t+1])

end;

{\* 若是第9操作那就判断大于值 \*}

12: begin

t := t-1;

s[t] := ord(s[t] > s[t+1])

end;

{\* 若是第9操作那就判断小于等于值 \*}

13: begin

t := t-1;

s[t] := ord(s[t] <= s[t+1])

end;

end;

{\* 若f是lod \*}

lod : begin

t := t+1;

s[t] := s[base(l)+a]

end;

{\* 若f是sto \*}

sto : begin

s[base(l)+a] := s[t]; { writeln(s[t]); }

t := t-1

end;

{\* 若f是cal \*}

cal : begin { generate new block mark }

s[t+1] := base(l);

s[t+2] := b;

s[t+3] := p;

b := t+1;

p := a;

end;

{\* 若f是int \*}

int : t := t+a;

{\* 若f是jmp \*}

jmp : p := a;

{\* 若f是jpc \*}

jpc : begin

if s[t] = 0

then p := a;

t := t-1;

end;

{\* 若f是red \*}

red : begin

writeln('??:');

readln(s[base(l)+a]);

end;

{\* 若f是wrt \*}

wrt : begin

writeln(s[t]);

t := t+1

end

end { with,case }

until p = 0;

writeln('END PL/0');

end; { interpret }

begin { main }

writeln('please input source program file name : ');

{\* 让用户输入Source program file名放在sfile变量 \*}

readln(sfile);

{\* 让fin指向sfile \*}

assign(fin,sfile);

reset(fin);

for ch := 'A' to ';' do

ssym[ch] := nul;

{\* 局部变量定义 \*}

word[1] := 'begin '; word[2] := 'call ';

word[3] := 'const '; word[4] := 'do ';

word[5] := 'end '; word[6] := 'if ';

word[7] := 'odd '; word[8] := 'procedure ';

word[9] := 'read '; word[10]:= 'then ';

word[11]:= 'var '; word[12]:= 'while ';

word[13]:= 'write ';

wsym[1] := beginsym; wsym[2] := callsym;

wsym[3] := constsym; wsym[4] := dosym;

wsym[5] := endsym; wsym[6] := ifsym;

wsym[7] := oddsym; wsym[8] := procsym;

wsym[9] := readsym; wsym[10]:= thensym;

wsym[11]:= varsym; wsym[12]:= whilesym;

wsym[13]:= writesym;

ssym['+'] := plus; ssym['-'] := minus;

ssym['\*'] := times; ssym['/'] := slash;

ssym['('] := lparen; ssym[')'] := rparen;

ssym['='] := eql; ssym[','] := comma;

ssym['.'] := period;

ssym['<'] := lss; ssym['>'] := gtr;

ssym[';'] := semicolon;

mnemonic[lit] := 'LIT '; mnemonic[opr] := 'OPR ';

mnemonic[lod] := 'LOD '; mnemonic[sto] := 'STO ';

mnemonic[cal] := 'CAL '; mnemonic[int] := 'INT ';

mnemonic[jmp] := 'JMP '; mnemonic[jpc] := 'JPC ';

mnemonic[red] := 'RED '; mnemonic[wrt] := 'WRT ';

declbegsys := [ constsym, varsym, procsym ];

statbegsys := [ beginsym, callsym, ifsym, whilesym];

facbegsys := [ ident, number, lparen ];

err := 0;

cc := 0;

cx := 0;

ll := 0;

ch := ' ';

kk := al;

{\* 读下一个符号 \*}

getsym;

block( 0,0,[period]+declbegsys+statbegsys );

{\* 若sym不是句号，那就是第9错误 \*}

if sym <> period

then error(9);

{\* 若err是0，执行interpret \*}

if err = 0

then interpret

else write('ERRORS IN PL/0 PROGRAM');

writeln;

close(fin)

end.